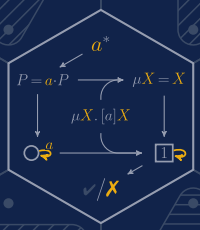
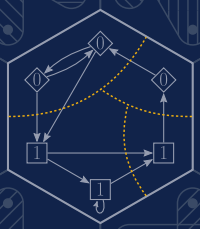
Reductions for Parity Games and Model Checking

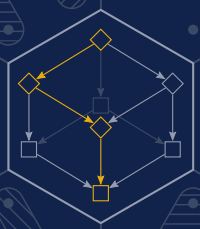
Thomas Neele

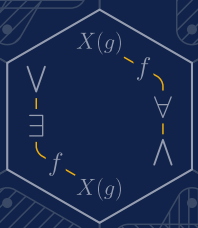
Eindhoven University of Technology 16-09-2020

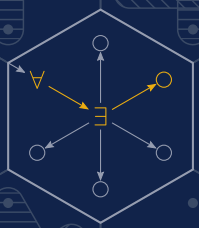
Reductions for Parity Games and Model Checking











Thomas Neele

Reductions for Parity Games and Model Checking

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 16 september 2020 om 16:00 uur

door

Thomas Sebastiaan Neele

geboren te De Bilt

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. M.T. de Berg
1e promotor:	dr.ir. T.A.C. Willemse
2e promotor:	prof.dr.ir. J.F. Groote
leden:	prof.dr. W.R. Cleaveland (University of Maryland)
	prof.dr. W.J. Fokkink
	dr. R. Mateescu (INRIA Grenoble - Rhône-Alpes)
	prof.dr. M. Huisman (Universiteit Twente)
	prof.dr. A. Valmari (University of Jyväskylä)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The work in this thesis has been carried out as part of the IMPULS II program. The position of the author was co-funded by Eindhoven University of Technology and ASML.

IPA dissertation series 2020-07.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5089-0

Printed by ProefschriftMaken.

Cover: 157 of the 203 possible partitions of a six element set and one cut-out visualisation of each of the six main chapters.

Preface

My career as a young scientist perhaps started on the day that I attended a meeting between several researchers, among others Marieke Huisman, from the Universiteit Twente (UT) and Anton Wijs from the Technische Universiteit Eindhoven (TU/e). At the time, I was working on my bachelor thesis, and this was the first time I witnessed science from the inside. During the next two years, though, I was still convinced that I wanted to find a job in industry. This did not significantly change during my three month internship at the Institute of Software, Chinese Academy of Sciences (ISCAS), which was made possible by the great support of Lijun Zhang. However, on my first day in Eindhoven, when I had just started working on my master thesis under the supervision of Anton Wijs, the PhD students who were around immediately tried to convince me to do a PhD. This did not have any effect until I started realising how rewarding it is to do (successful) research, and so I asked Jan Friso Groote about the available positions.

One thing led to another, and now I find myself with a completed thesis, which I could not have managed alone. Therefore, I want to thank several people who helped me along the way. First of all, my gratitude goes out to the people who supported me during my time as a bachelor and master student: Marieke Huisman, Jaco van de Pol, Lijun Zhang, Anton Wijs and Dragan Bošnački. You have all been a great example to me, and helped me discover the beauty of computer science.

Next, I want to thank Jan Friso Groote and Tim Willemse, my promotors. I will start off, however, by giving some insight into how my relation with each of them developed throughout my PhD. Since Jan Friso was responsible for creating my position, he naturally acted as my supervisor from the beginning. We had weekly meetings on Wednesdays, when we saw each other at ASML. Our initial aim was to improve the ability to model check real-time systems in mCRL2, perhaps building on the work that he started with the tool `lpsrealelm`. I worked on this for several months, first creating the tool `lpsrealzone`, for reasoning about timed automata with zones, and later `lpssymbolicbisim`. Since we were not satisfied with their scalability (Jan Friso: "we should be able check models with more than six trains!"), I went looking for alternatives and ended up porting the symbolic technique to

parameterised Boolean equations systems (PBESs). This was no coincidence, since PBESs are the favourite formalism of Tim, who acted as my co-supervisor. The magic power of PBESs also captured me, so after finishing our paper on symbolic PBES solving, I found myself working on quantifier manipulation techniques for PBESs and partial-order reduction for PBESs. Naturally, this meant that I was working more closely with Tim during the second half of my PhD.

Jan Friso, since you once told me "I am not supervising you, we are collaborating", I will use your words and say: thanks for the incredible collaboration. Even though I have always been ambitious, you reinforced my strive for excellence. Your desire to make real steps forward motivated me to focus more on theory and less on small optimisations that are irrelevant on the grander scale. The time you took to discuss topics not related to my research helped me gain valuable insights into the inner workings of the academic world.

Tim, when you asked me to serve as a co-supervisor and have weekly meetings, I did not really understand the purpose, since I thought I would mostly be working under Jan Friso anyway (now I know it is very common to have multiple supervisors). Looking back, I am very glad I accepted your offer: your supervision has been nothing but outstanding: you always guided me in the right direction. During our weekly discussions, you were always quick to understand the problems I was facing. The time you took to play around with our experimental tools resulted in valuable insights and the writing you contributed to our papers significantly improved their presentation. Your positivity and sense of humour helped a great deal whenever the results of the research were disappointing.

There are also several other people who directly contributed to my thesis in some way. Firstly, I want to thank my co-authors Wieger Wesselink, for helping with the implementation of several ideas in the mCRL2 tool set, and Antti Valmari, for providing the sharpest feedback I could wish for. The two chapters on partial-order reduction would not have been the same without your help. To the members of my committee, Rance Cleaveland, Wan Fokkink, Radu Mateescu, Marieke Huisman and Antti Valmari, thanks for devoting your precious time to read the result of my work. Your comments have helped to resolve the remaining issues in the text and in the formalisations.

My thanks goes out to ASML and the people involved in the IMPULS II/ASOME^2 project. My fellow PhDs in the project, Kousar Aslam, Ruben Jonk and Nan Yang, you were great office mates and board game rivals. Ramon Schiffelers, thank you for giving me the opportunity to follow my own path and focus on fundamental research; I truly enjoyed the freedom that I had during my PhD. The meta-discussions I had with Jeroen Voeten were valuable to me, and changed my perspective on the academic world; thanks!

As a PhD candidate, I was responsible for helping with two courses: Design Based Learning Embedded Systems and Automotive Software Engineering. I learned a great deal myself, thanks to my colleagues Pieter Cuijpers, Erik de Vink, Jan Friso Groote, Tim Willemse and Jeroen Keiren.

In the Formal Systems Analysis (FSA) group, the mCRL2 model checking toolset

is a great foundation for research activities. The fact that many basic algorithms were already provided saved me a lot of time implementing my own ideas. I enjoyed working on several parts of the toolset in a development team where everybody had their own approach to development. Jan Friso Groote, the main driving force behind the toolset for over 15 years, always manages to surprise by producing enormous amounts of code over the weekend, if he thinks a certain feature really should be implemented. The most beautiful code and documentation comes from the hand of Wieger Wesselink, who set an example for all of us. Maurice Laveaux's knowledge of C++ and OpenGL led to many fundamental improvements in the toolset, including a complete rewrite of the ATerm library and its binary storage format and also a refactoring of `ltsgraph`. Olav Bunte deserves a lot of credit for single-handedly creating the user-friendly `mCRL2ide` tool, which made my job teaching mCRL2 that much easier. Ferry Timmers has a great eye for detail and his first contributions have been the introduction of often-requested features to `ltsgraph`. Our unofficial tester is Tim Willemse, who always managed the find the most curious bugs.

My time at the TU/e would not have been as enjoyable without all the fellow PhDs in the Model-Driven Software Engineering (MDSE) cluster. I am truly grateful to Sander de Putter and Mahmoud Talebi, who welcomed me into the group when I was working on my master thesis; you really made me feel at home in Eindhoven. Out of all the things we have done together, our trip to Iran was a most memorable experience, not least witnessed by all the food we got to try. Weslley Silva Torres, you were single-handedly responsible for making the office as lively as it could be. The fact that we disagreed on so many topics always created the most interesting discussions at the lunch table. But above all you are a great friend; you even entrusted some of the organisation of your wedding to me. I hope you will enjoy life in the Netherlands together with Giovanni Calheiros. Omar al Duhaiby, thanks for all the interesting discussions, both about science and about life. Felipe Ebert and Camila Kokkosi, thanks for the amazing barbecues where we ate picanha with farofa and drank guaraná. Priyanka Karkhanis and Markus Klinik, you were always eager to join activities; we had some great fun during various Friday afternoon drinks and random dinners and certainly during the Christmas parties.

Amazingly, I also did sports once in a while, and some of my colleagues even managed to involve me in a couple of new sports. Dan Zhang, thanks for motivating me to try BBB and joining me several times. Rick Erkens, your passion for weightlifting is inspiring. We must have done something wrong the first time, since the second time we tried to go to the gym, the door was locked. In the last year of my PhD, I discovered bouldering through Nathan Cassee and Lars van den Haak. Thanks for teaching me the beginnings!

To my (former) office mates, Alexander Fedotov, Mauricio Verano Merino, Maurice Laveaux, Olav Bunte, Muhammad Osama, Ruud van Vijfeijken and Jan Martens, thanks for making it such a pleasant workspace. I would also like to thank all the other people in MDSE that spent time with me: Fei Yang, Rodin Aarssen, Sarmen Keshishzadeh, Ferry Timmers, Mark Bouwman, Sangeeth Kochanthara and Ayushi Rastogi, Yaping (Luna) Luo and Valcho Dimitrov, Önder Babur, Kousar Aslam, Nan

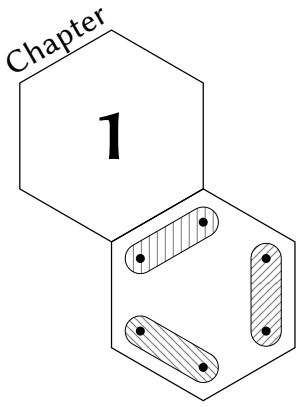
Yang, Miguel Botto Tobar, Mahdi Saeedi Nikoo, Ana-Maria and Alex Şutîi, Ulyana Tikhonova, Yanja Dajsuren, Gema Rodriguez Perez, Lina Ochoa, Tukaram Muske, Raquel Álvárez Ramirez, Josh Mengerink, Arash Khabbaz Saberi, Jouke Stoel and Yuexu (Celine) Chen. All the experiences we had together made my PhD time truly enjoyable.

There are also several important people outside the university that I want to thank. Saurab Rajkarnikar and Megha Vaidya, thanks for inviting Bulgaa and me regularly and cooking your delicious momos. Tamir Tsedenjav and Bayasgalan Baatar, thanks for welcoming me into your home, even during the hardest times. Robert van de Vlasakker, thanks for all the great memories (lasagnebadminton etc.). I am happy that we are still in contact.

Finally, the time has come to thank my family. First of all, to my brother Laurens, who is an inexhaustible source of trivia, thanks for being the best companion, especially during the time we lived together in Eindhoven. My parents, Jos and Margreet, supported my never-ending curiosity during my childhood and continued encouraging me when I entered university and later started my scientific career. I owe them for giving me the best upbringing I could wish for. Bulgaa, my love, thanks for sharing your life with me. We will soon embark on a new adventure together, and I could wish for no better partner to share it with.

Thomas Neele, July 2020

Contents



Introduction

One of the major inventions of the twentieth century is the electronic computer. Running a computer requires loading it with a program that it can execute, also called *software*. The task of designing and implementing software programs is called *programming*. Already in the early days of computing, the problem of writing high-quality software that meets all expectations was identified [1, 39]. This is known as the *software crisis*. Its consequences can be manyfold: software projects run over time and budget and bugs in end products such as cars, air planes and medical devices can endanger lives. Since the amount of software is increasing rapidly [40], the software crisis has a growing impact on society.

In the past half century, many different approaches to tackling the software crisis have been proposed. These range from the use of high-level programming languages, *integrated development environments* with many programmer aides and software development paradigms such as *agile* to intensive testing with *continuous integration*. Although the latter can address some of the issues [134], the problem is far from solved.

In modern day industry, software testing is still the most popular way of validating whether a given piece of software meets its intended purpose [120]. However, testing is very much an empirical approach: the software is executed a number of times, subject to different scenarios. Afterwards, one checks whether the results meet the

expectations. For most software, only an infinite number of test runs, each with a different input, can give absolute certainty about its correctness.

These issues are compounded when developing multi-threaded software. Tests of single-threaded software are largely reproducible: two runs with the same input will most likely result in the same output. However, the behaviour of multi-threaded software can depend on the order in which the threads are scheduled. This ordering is influenced by many external factors, which may include hardware performance and current system load. As a result, multi-threaded software can seemingly behave non-deterministically, resulting in a different result every test run. This makes the probability of finding all bugs in concurrent software close to zero. For such software, testing is thus an inadequate method of establishing the correctness.

1.1 Formal Methods

Whereas testing is an empirical process, formal methods rely on theory to *prove* that a given piece of software is correct. This is achieved by mathematically reasoning about behaviour of software. Such an approach is more rigorous than testing: it does not depend on the creativity of the test engineer to come up with scenarios that require testing or the number of runs that one performs, but it checks all possible scenarios. In this way, formal methods provide a higher degree of certainty on the correctness of software.

Reasoning about the behaviour of software is often based on a definition of its *semantics*: a set of mathematical rules that precisely define what the effect of each element of a program is. We highlight several different types of semantics and techniques that apply them [6]. Firstly, techniques that use *axiomatic* semantics reason, based on certain axioms and proof rules, what is true before and after each program statement. This includes deductive verification [45, 61] with Hoare logic or separation logic. Secondly, in *denotational* semantics, every program is represented by a corresponding abstract object. For simple programs, these objects can take the shape of a function from input to output. Abstract interpretation [32] is a technique that employs denotational semantics. Lastly, there are techniques based on *operational semantics*, where the behaviour of a program is typically represented as a directed graph. For every state of the program, this graph contains a node; the edges indicate possible changes in the program state. This thesis focusses on *model checking* techniques that are based on operational semantics.

1.2 Model Checking

Model checking [9] is an automated technique for establishing whether certain properties hold for a given system. The behaviour of the system under consideration is typically modelled by a *specification*, which is a compact representation of a *transition system*, in the form of a directed graph. Common examples of specification formalisms

are *process algebras* [7, 98], *Petri nets* [113], *timed* [4] and *hybrid automata* [3] and various kinds of probabilistic models, such as *Markov chains* [94]. The formal properties are usually given as a formula in some temporal logic, such as *linear temporal logic* (LTL) [116], *computation tree logic* (CTL) [30] or the *modal μ-calculus* [83]. Apart from software systems, model checking can also be applied to many other types of systems, such as logic circuits, network protocols, planning problems, board games and critical infrastructure like railroads.

The most straightforward model checking procedures take a specification and iteratively generate the underlying transition system, starting from the initial state. This process is commonly known as *state-space exploration*. Evaluating the temporal formula on the resulting transition system answers the question whether the specification satisfies the property under consideration. Sometimes this evaluation can even be performed *on-the-fly*, *i.e.*, during the exploration process.

A fundamental problem in model checking is the size of the state space, which tends to be very large. In a system specification containing multiple concurrent processes, the independent behaviour of two or more processes can be *interleaved* in an arbitrary order. Consequently, any combination of states of these processes is a state of the system as a whole. A linear increase in the number of concurrent processes thus leads to a combinatorial (exponential) increase in the size of the state space. This phenomenon is commonly known as the *state-explosion problem* [129]. Moreover, if a model contains data of an infinite domain and does not restrict its values, the state space also becomes infinite. Timed automata are a prime example: their real-valued clocks cause the state space size to be uncountably large. Such formalisms often require specialised model checking algorithms or abstraction methods.

Several possible solutions have been proposed in the literature to address these issues, such as various symbolic model checking techniques (*e.g.* model checking with BDDs [25, 27, 89] and bounded model checking [19]), abstraction methods (*e.g.* counter-example guided abstraction refinement [29] and time-abstracting bisimulation [124]) and techniques that specifically address the arbitrary interleaving of independent behaviour (*e.g.* symmetry reduction [67] and partial-order reduction [52, 111, 127]).

Many of these techniques apply to specifications only and do not take any knowledge of the property into account. This limits the reduction potential, which can only be fully exploited by also considering the property [96]. Other techniques support only (a fragment of) LTL or CTL, and cannot be used when one wants to check a property in a more expressive logic, such as the μ-calculus.

1.3 Parameterised Boolean Equation Systems

The logical framework of *parameterised Boolean equation systems* (PBESs) [55, 59, 95] can partially resolve these issues. Apart from model checking queries [55], PBESs can encode many different types of decision problems, such as the question whether two specifications are related according to some behavioural equivalence/preorder [28],

software verification problems [80], satisfiability of μ-calculus formulae [24] and several string problems [66]. *Solving* a PBES yields the answer to the decision problem it encodes. It should be noted, though, that the problem of solving a PBES is in general undecidable.

Similar to how a behavioural specification compactly represents a transition system, a PBES abstractly represents a directed graph, typically in the form of a *parity game* [43, 97]. A common way of solving a PBES, is to *instantiate* its underlying parity game through a process similar to state-space exploration [114]. The parity game can be solved with one of many solving algorithms from literature, such as Zielonka's recursive algorithm [140]. Parity game solving is one of a few problems which are in *UP* and *co-UP*, but not known to be in *P*. The existence of an algorithm that can solve a parity game in polynomial time is a major open problem.

The versatility of PBESs and their relation to game theory alone justify further study of PBESs and how to solve them. In [59], Groote and Willemse envision PBESs to become a universal framework that can encode many different types of problems from theoretical computer science. In this way, they can fulfil the same role that differential equations have in other engineering disciplines.

However, PBESs by no means resolve the state-explosion problem in model checking, since the encoded parity games also grow exponentially with the number of concurrent processes. Therefore, we need similar state space reduction techniques to combat the blow up. Since PBESs and parity games encode the combination of a specification and a property, any technique that we apply to them automatically considers both the specification and the property, potentially increasing the amount of reduction. We thus have the following aim:

> Improve existing PBES solving procedures by means of reduction techniques, and either speed up PBES solving or extend the class of PBESs that can be solved.

This ultimately improves the applicability of model checking and can assist in addressing some aspects of the software crisis. Furthermore, our improved PBES solving routines may be efficient (semi-)decision procedures for problems other than model checking.

The application of existing state space reduction techniques to PBESs and parity games is not trivial, however. Firstly, the predicate formulae contained in a PBES may have an arbitrary structure, making it difficult to statically determine which transitions exist in the parity game underlying a PBES. Secondly, it is not obvious that a state space reduction technique, which preserves a certain class of properties of a transition system, also preserves the solution of parity games that encode the same class of properties. Lastly, existing techniques do not automatically exploit all the reduction potential in a PBES; we may thus need to introduce new optimisations to achieve the best reduction.

1.4 Contributions

The main topic of this thesis is the study of three kinds of reduction techniques for PBESs and parity games: *PBES quotienting* for solving PBESs with an infinite underlying parity game, the application of *partial-order reduction* (POR) to PBESs and a number of syntactical transformations for PBESs that aim to reduce the underlying parity game. Furthermore, the thesis also discusses a correctness issue in existing partial-order reduction theory for the setting of behavioural specifications. These contributions are discussed in more detail below.

First, Chapter 2 introduces the notions that are required to understand the rest of the thesis. The main concepts introduced in Chapter 2 are *labelled transition systems*, *linear processes*, the *modal μ-calculus*, *parity games* and *parameterised Boolean equation systems*. We also show how these concepts relate to each other.

The main body of the thesis starts with investigating the issue that, although PBESs themselves offer a clear structure, the predicate formulae contained in PBESs can take any form, complicating the analysis of dependencies within a PBES. PBES dependencies roughly correspond to parity game transitions. Hence, precise knowledge of dependencies is essential for the application of many state space reduction techniques. Although previous works show how to capture some of this knowledge in so called *dependency graphs* [37], these cannot capture both positive dependencies and negative dependencies at the same time. Therefore, we pose the following research question:

> **RQ1** What is an effective way of obtaining all dependencies within a PBES?

The solution proposed in Chapter 3 consists of two new normal forms for PBESs: *standard recursive form* and *clustered recursive form*. These normal forms offer the structure that ordinary PBESs lack and enable one to capture all (positive and negative) dependencies in a structure called *dependency space*. The ideas of Chapter 3 are fundamental for the techniques proposed in subsequent chapters.

Next, we discuss symbolic techniques for PBESs that have an infinite underlying parity game. Although several procedures to deal with this type of PBES have been proposed [81, 100], they can only handle a fragment of the PBES logic, due to a lack of the necessary normal form. This limits their applicability to checking linear-time (LTL) properties. Hence, we ask ourselves whether these ideas can be generalised:

> **RQ2** Is it possible to perform automated infinite-state model checking of arbitrary μ-calculus properties with PBESs?

By leveraging the normal forms from Chapter 3, Chapter 4 demonstrates how to perform *PBES quotienting* on arbitrary PBESs, which takes ideas from *minimal model generation* [22] for transition systems. Furthermore, the setting of PBESs allows two more optimisations that are not possible in the traditional setting of transition systems. Experiments with an implementation of PBES quotienting show

that these ideas work well for several model checking and equivalence checking examples. Our tool is, to our knowledge, the first to implement a semi-decision procedure for bisimilarity checking of infinite systems.

We continue with a discussion on partial-order reduction, a technique that aims to reduce the number of interleavings (and hence the number of states) explored by prioritising the behaviour of some process whenever this does not impact which sequences of labels are observed. Although there are many different variants of POR, among which are *ample sets* [111], *persistent sets* [52] and *stubborn sets* [127], a common theme is that they all reason about actions, which label transitions, while they aim to preserve all sequences of state labels. Due to this disparity, correctness of POR is not obvious, and we investigate the following:

> **RQ3** What are the correctness implications of the way POR deals with actions and state labels?

Chapter 5 shows that two early works [126, 128], which propose a technique for stubborn sets to preserve LTL without the *next* operator, do not properly deal with the distinction between actions and state labels. As a consequence, the application of these ideas in state-space exploration may yield a transition system that satisfies different properties than the original transition system captured in the specification. We refer to this as the *inconsistent labelling problem*. We propose updated stubborn set conditions that resolve the issue and discuss multiple related works that are affected. The impact on most practical implementations is limited, since they compute an approximation of stubborn sets.

After we established a correct POR theory for the setting of transition systems, we subsequently investigate its application to parity games. POR has seen many successful applications in the past, but most approaches share the limitation that they at best preserve CTL without the next operator. Designing a POR technique that operates on parity games and that preserves the winner in any given game would resolve this issue. This invites the following long standing open question [73, 114, 115, 138]:

> **RQ4** How can partial-order reduction be applied to PBESs and parity games?

The approach presented in Chapter 6 builds on the corrected conditions of Chapter 5 and leverages the standard recursive form for PBESs (Chapter 3). This results in the first POR technique that can be applied while checking any μ-calculus formula, including stutter-sensitive formulae. Experiments indicate that, for non-trivial examples, substantial reductions can be achieved.

The last topic we discuss are syntactical transformations for PBESs that aim to reduce the effort spent in solving them. One possible reason for a PBES to have a large underlying parity game is the occurrence of quantifiers whose bound variable can take arbitrary values. These typically occur when one wants to check a certain property for multiple (similar) processes in a specification. Existing static

analysis techniques [106, 74] deal poorly with quantifiers. Thus, there may be a lot of unexploited reduction potential, and we consider the following question:

> **RQ5** How can quantifiers in a PBES be manipulated such that the underlying parity game is smaller?

Chapter 7 proposes two possible techniques, called *quantifier propagation* and *global propagation*. The latter is a completely automated procedure, which ensures quantifiers only occur in those places where the value selected for its variable is relevant. Global propagation generalises the constant elimination algorithm from [106].

The same chapter also considers the concept of *guards*, which are a useful tool in most static analysis techniques. A guard is an expression that characterises a dependency within a PBES. However, the guards computed by [74] over-approximate those dependencies. If we can compute more precise guards, this will support more powerful static analysis. Hence, we investigate:

> **RQ6** Can we efficiently compute more precise guards or even exact guards?

The first half of this question is answered affirmatively: the computation presented in Section 7.6 computes stronger guards than existing works. We prove that the guards we compute are *compositional*: strengthening a formula with a guard does not change other guards. However, to answer the second half of the question, we demonstrate that exact guards, which exactly characterise PBES dependencies, do not have this desirable property. As a result, computing and applying exact guards is infeasible for large PBESs.

To conclude the thesis, we review each of the proposed techniques and we try to answer the following question:

> **RQ7** What are the advantages and disadvantages of the application of existing reduction techniques to PBESs?

The main advantage discussed in Chapter 8 is the fact that PBESs often facilitate larger theoretical reductions. However, most information related to the individual processes is lost, which can make static analysis challenging in practice. Chapter 8 ends with suggestions for future work.

1.5 Origin of the Chapters

The main content of the thesis is divided up into three parts which can mostly be read independently. The only interdependence is the introduction of the concept of *standard recursive form* in Chapter 3 and its use in Chapter 6.

The first part contains Chapters 3 and 4, which both originate from the following two publications.

[103] T. Neele, T. A. C. Willemse, and J. F. Groote, Solving Parameterised Boolean Equation Systems with Infinite Data Through Quotienting. In *FACS 2018*, vol. 11222 of *LNCS*, pp. 216–236, 2018. **Best paper award.**

[104] T. Neele, T. A. C. Willemse, and J. F. Groote, Finding Compact Proofs for Infinite-Data Parameterised Boolean Equation Systems. *Science of Computer Programming (FACS 2018 special issue)*, vol. 188, 102389, 2020.

The latter publication extends the former with an explanation of minimal model generation, a larger running example, full correctness proofs and a more comprehensive experimental evaluation. The theory of these papers is completely based on dependency graphs and proof graphs. For consistency, the thesis presents the same ideas in terms of parity games. The contributions of [103] were recognised with the FACS 2018 best paper award.

The second part of the thesis consists of Chapters 5 and 6, which respectively originate from the following publications.

[102] T. Neele, A. Valmari, T. A. C. Willemse, The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction. In *FoSSaCS 2020*, vol. 12077 of *LNCS*, pp. 482–501, 2020. **Twice nominated for best paper award; won EATCS best paper award.**

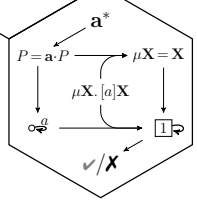
[105] T. Neele, T. A. C. Willemse, W. Wesselink, Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems. In *TACAS 2020*, vol. 12079 of *LNCS*, pp. 307–324, 2020.

The observations presented in Chapter 5 originated while developing the theory of Chapter 6. In the thesis, Chapter 6 is presented as an application of (part of) the fundamental theory in Chapter 5. Chapter 5 is largely independent of the preliminaries presented in Chapter 2; it only relies on *labelled transition sytems* and its related concepts presented in Section 2.2. The main counter-example in Chapter 5 was contributed by Antti Valmari and the tool used in the experiments of Chapter 6 was implemented with the help of Wieger Wesselink. The paper [102] received the EATCS award for the best ETAPS 2020 paper in theoretical computer science and was also nominated for the EASST award for the best ETAPS 2020 paper related to the systematic and rigorous engineering of software and systems.

Chapter 7 forms the last part of the thesis. This chapter is unpublished.

Chapter

2



Preliminaries

This chapter introduces basic notions that are used throughout the thesis. These include abstract data, *labelled transition systems* as a basic representation of behaviour and *parity games* to encode several types of decision problems on transition systems. Furthermore, we introduce *linear processes* and *parameterised Boolean equation systems* as compact representations of transition systems and parity games, respectively. To express formal properties of an LTS, we use the *modal μ-calculus*.

2.1 Abstract Data

Throughout the thesis, we work with abstract data types and expressions over those data types. We distinguish their syntax and semantics. Data sorts are denoted with the letters $D, E, \ldots$. The corresponding semantic domains are $\mathbb{D}, \mathbb{E}, \ldots$; we assume that these semantic domains are non-empty. In addition, we use B to denote the Booleans and N to denote the natural numbers $\{0, 1, 2, \ldots\}$, which have the semantic counterparts $\mathbb{B}$ and $\mathbb{N}$ respectively. Booleans and natural numbers are primarily used in the examples. We also have a singleton sort $D_\star = \{\star\}$ on which no operations are defined. Furthermore, we have a set of data variables $\mathcal{V}$. In syntax, variables are typically denoted with the letters d and e, while semantic values are denoted with v and w. To indicate that the type of variable d is D, we write $d{:}D$. Expressions not

containing variables are called *ground terms*.

To interpret expressions that contain variables, we have a *data environment* δ that maps each variable in $\mathcal{V}$ to an element of the corresponding domain. We use $[\![\cdot]\!]$ as the interpretation function, *i.e.*, the semantics of an expression f in the context of a data environment δ is denoted $[\![f]\!]\delta$. If f is a ground term, we may also write $[\![f]\!]$, since the data environment is not relevant for the semantics of f. The set of all data environments is Δ. Updates to an environment δ are denoted by $\delta[v/d]$, which is defined as $\delta[v/d](d) = v$ and $\delta[v/d](d') = \delta(d')$ for all variables d, d' satisfying $d' \neq d$.

We sometimes write $f(d)$ to emphasise that the expression f only depends on d. Remark that the value of $[\![f(d)]\!]\delta[v/d]$ does not depend on δ. For those cases, we assume the existence of some fixed data environment δ_0, and write $[\![f(d)]\!]\delta_0[v/d]$.

Example 2.1. Let m and n be variables of type N and δ a data environment. We consider the value $v = [\![m(n+1)]\!]\delta[6/n]$. If $\delta(m) = 2$, we have $v = 14$. □

2.2 Transition Systems and Processes

The atomic elements of behaviour that we consider are *actions*, typically denoted with a. Each action represents an event in the real world, such as "a key is pressed" or "the variable x is set to 1". Thus, the occurrence of an action most often coincides with a change in the state of the system that we are considering. The relation between actions and system states is captured in a *labelled transition system* (LTS) [76]. This is a possibly infinite, directed graph where the edges are labelled with actions. Here, we assume the existence of a fixed set of actions Act, which may be infinite.

Definition 2.2. A *labelled transition system* (LTS) is a three-tuple $TS = (S, \rightarrow, \hat{s})$, where

- S is a set of states, which we refer to as the *state space*;
- $\rightarrow \subseteq S \times Act \times S$ is the transition relation; and
- $\hat{s} \in S$ is the initial state.

Below, we will use the terms LTS and transition system interchangeably. We write $s \xrightarrow{a} t$ whenever $(s, a, t) \in \rightarrow$. An action a is *enabled* in a state s, notation $s \xrightarrow{a}$, iff there exists a state t such that $s \xrightarrow{a} t$. Given an LTS TS, the set of all enabled actions in a state s is denoted $\mathit{enabled}_{TS}(s)$. We call a state s a *deadlock* iff $\mathit{enabled}_{TS}(s) = \emptyset$. A *path* is a (finite or infinite) alternating sequence of states and actions: $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$. We sometimes omit the intermediate and/or final states if they are clear from the context or not relevant, and write $s \xrightarrow{a_1 \ldots a_n} t$ or $s \xrightarrow{a_1 \ldots a_n}$ for finite paths and $s \xrightarrow{a_1 a_2 \cdots}$ for infinite paths. A path that starts in the initial state $\hat{s}$ is called an *initial path*.

We say a state s is *deterministic* if and only if $s \xrightarrow{a} t$ and $s \xrightarrow{a} t'$ imply $t = t'$, for all states t and t' and actions a. An LTS is deterministic iff all its states are

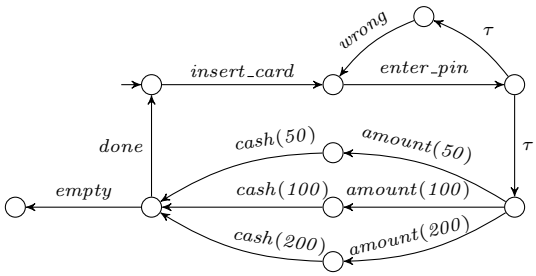


Figure 2.1: An LTS modelling an ATM. Each circle represents a state and arrows represent transitions. The initial state $\hat{s}$ is indicated with an incoming arrow.

deterministic. We call the occurrence of a state s and an action a such that $s \xrightarrow{a} t$ and $s \xrightarrow{a} t'$ where $t \neq t'$ a *non-deterministic choice*. Non-determinism is often used as an abstraction method: instead of precisely modelling a complex (possibly unknown) process that can have multiple outcomes, one can abstractly represent the choice that the process makes with non-determinism.

To make the ideas more concrete, we present an example of a small LTS. This system will serve as a running example throughout the current chapter.

Example 2.3. We consider the LTS drawn in Figure 2.1, which models the behaviour of an ATM. States are depicted with circles and transitions with arrows. The initial state has an incoming arrow. After inserting a bank card and entering a PIN, the ATM decides whether the PIN was correct. Since we do not want to model the existence of different cards and their PINs, this process is modelled abstractly with a non-deterministic choice, represented by the action τ. Of course the actual ATM contains some internal routine to decide whether the PIN was correct. If the PIN is wrong, it has to be entered again; if the PIN is correct, one can choose an amount of cash to withdraw. Here, the action $cash(50)$ represents the machine ejecting 50 units of cash and returning the bank card. When the ATM is out of cash, it performs the action *empty*, which leads to a deadlock state. Otherwise, it returns to the initial state. □

Typically, one does not specify an LTS directly, but through a higher level modelling language. One family of such languages are *process algebras* [7]. Popular examples are CCS [98] and ACP [16]. In process algebra, actions are also considered as behavioural atoms. They can be combined with one or more operators to form complex specifications of behaviour, known as *processes*. Note that an action by itself is also a process. In this thesis, we restrict ourselves to *linear processes*. The combination of a linear process and an initial state is a *linear process specification* (LPS). In the definition, we use some domain D to represent the state of the system. Furthermore, actions consist of an *action label* and an *action parameter*, taken from some domain D_{par}, which represents some data that is associated with the action. For

example, in a process modelling a networking protocol, the action label *send_ack* may be accompanied by an action parameter that indicates which message is acknowledged.

Definition 2.4. A *linear process specification* (LPS) is a tuple $L = (P, \hat{d})$, where $\hat{d}$ is the initial state, given by a ground term of sort D, and P is a recursive process of the shape

$$P(d{:}D) = \sum_{i \in I} \sum_{e_i : E_i} \big(c_i(d, e_i) \to a_i(f_i(d, e_i)) \cdot P(g_i(d, e_i))\big)$$

where I is a finite (possibly empty) index set, e_i is a summation variable ranging over the non-empty domain E_i, c_i is a boolean condition, a_i is an action label that has the parameter $f_i(d, e_i)$ of type D_{par} and $g_i(d, e_i)$ is an expression of type D.

Intuitively, in every state represented by variable d, a linear process offers a (non-deterministic) choice to perform an action $a_i(f_i(d, e_i))$ that is enabled, *i.e.*, for which $c_i(d, e_i)$ evaluates to *true* for some value of e_i. After performing this action, the state is updated to $g_i(d, e_i)$. This is captured in the LTS that forms the semantics. In the definition below, the set of actions Act follows from the action labels and action parameters in the LPS, so we have $Act = A \times \mathbb{D}_{par}$ for some finite set A containing the action labels of the LPS.

Definition 2.5. Let $L = (P, \hat{d})$ be an LPS in the shape of Definition 2.4. Then the LTS associated with L is defined as $TS_L = (\mathbb{D}, \to, [\![\hat{d}]\!])$, where $\to$ is the set satisfying for all $v \in \mathbb{D}$, $i \in I$ and $v_i \in \mathbb{E}_i$, $(v, (a_i, [\![f_i(d, e_i)]\!]\delta_0[v/d, v_i/e_i]), [\![g_i(d, e_i)]\!]\delta_0[v/d, v_i/e_i]) \in \to$ if and only if $[\![c_i(d, e_i)]\!]\delta_0[v/d, v_i/e_i]$ holds.

Remark that in the LTS associated with an LPS, not all states in $\mathbb{D}$ are necessarily reachable from the initial state. In most of our analyses, we are only concerned with the reachable part of the state space.

Linear processes are a subset of the mCRL2 process algebra [56]. However, most mCRL2 processes can be translated into an LPS through a process called *linearisation* [125]. The structure of LPSs facilitates efficient state space generation and also other transformations. Contrarily, reasoning about mCRL2 processes with an arbitrary structure, *e.g.*, determining which actions are enabled, can be complex.

The assumption that a linear process has only one state parameter d of type D does not impact generality. After all, the corresponding semantic domain $\mathbb{D}$ can always be chosen such that $\mathbb{D} = \{\bot\} \cup \mathbb{D}_1 \cup (\mathbb{D}_2 \times \mathbb{D}_3) \cup \ldots$. The same applies to action parameters, which come from a single domain D_{par}. In the examples, we regularly consider LPSs with multiple state parameters and LPSs or LTSs where action labels carry a different number of parameters, or no parameter at all. Furthermore, in LPSs, we also unfold the first sum operator by using the *choice operator* (notation $+$) and we omit the second sum operator when e_i is not used.

Example 2.6. We revisit the LTS of Example 2.3. This LTS can be modelled with the LPS $(ATM, (idle, 0))$, where ATM is the following linear process.

$$
\begin{aligned}
&ATM(s{:}State, n{:}N) = \\
&\qquad (s = idle) \rightarrow insert\_card \cdot ATM(await\_pin, n) && (1)\\
&\quad + (s = await\_pin) \rightarrow enter\_pin \cdot ATM(check\_pin, n) && (2)\\
&\quad + \sum_{b:B} (s = check\_pin) \rightarrow \tau \cdot ATM(if(b, await\_amount, wrong\_pin), n) && (3)\\
&\quad + (s = wrong\_pin) \rightarrow wrong \cdot ATM(await\_pin, n) && (4)\\
&\quad + \sum_{m:N} (s = await\_amount \wedge (m = 50 \vee m = 100 \vee m = 200)) && (5)\\
&\qquad\qquad \rightarrow amount(m) \cdot ATM(deliver\_cash, m)\\
&\quad + (s = deliver\_cash) \rightarrow cash(n) \cdot ATM(check\_cash, 0) && (6)\\
&\quad + (s = check\_cash) \rightarrow empty \cdot ATM(no\_cash, n) && (7)\\
&\quad + (s = check\_cash) \rightarrow done \cdot ATM(idle, n) && (8)
\end{aligned}
$$

In this linear process, the parameter s stores the current control state of the ATM; its sort *State* can take the values *idle*, *await_pin*, *check_pin*, *wrong_pin*, *await_amount*, *deliver_cash*, *check_cash* and *no_cash*. The amount of cash that is to be withdrawn is stored in parameter n, which is a natural number. Note that the value n is only relevant just after the amount has been entered; after delivering the cash, it is reset to 0 and not used any more until another customer enters an amount.

To decide which actions can occur, we check the current value of s. On line 5, we allow one of three amounts to be entered by restricting the value of sum variable m in the condition. The non-deterministic choice whether to accept the PIN is modelled by introducing a Boolean sum variable b (line 3), whose value is not restricted by the condition. The value of b determines what the next state is, as per the expression $if(b, await\_amount, wrong\_pin)$.

Although we aimed to model a finite LTS, the LTS TS that is associated with $(ATM, (idle, 0))$ is infinite. However, the reachable part of TS exactly coincides with the LTS of Example 2.3. An example of an unreachable state in TS is $(await\_pin, 21)$. □

2.3 Modal μ-calculus

To express formal properties, we rely on the first-order modal μ-calculus [55], an expressive logic which finds its roots in Hennessy-Milner logic (HML) [64]. HML is a multi-modal logic; for each action a, there are two modal operators: the *diamond*, or "possibly", operator ($\langle a \rangle$) and the *box*, or "necessarily", operator ($[a]$). The μ-calculus [83, 84] extends this with *fixpoint* operators, which allow distinguishing between finite or infinite behaviour. Finally, the first-order μ-calculus [55] introduces

data; this enables specifying properties that involve data and infinite sets of actions. The expressiveness of the modal μ-calculus is due to the fact that all operators, including modalities and fixpoints, can be nested arbitrarily.

Before we formalise the μ-calculus itself, we first consider *action formulas*, which provide a syntactic way of specifying, possibly infinite, sets of actions. Here, and in the rest of this section, we again assume that each action is comprised of a label and a parameter, *i.e.*, Act is of the shape $A \times \mathbb{D}_{par}$.

Definition 2.7. *Action formulas* are generated by the following grammar:

$$\alpha, \beta ::= \; a(f) \mid \overline{\alpha} \mid \bot \mid \top \mid \alpha \cap \beta \mid \alpha \cup \beta \mid \sqcap_{d:D} \alpha \mid \sqcup_{d:D} \alpha$$

where a is an action label and f is a term of type D_{par}. The interpretation of an action formula α under a data environment δ, notation $[\![\alpha]\!]\delta$, is a set of actions. It follows the inductive definition:

$$\begin{aligned}
[\![a(f)]\!]\delta &= \{(a, [\![f]\!]\delta)\} & [\![\overline{\alpha}]\!]\delta &= Act \setminus [\![\alpha]\!]\delta \\
[\![\bot]\!]\delta &= \emptyset & [\![\top]\!]\delta &= Act \\
[\![\alpha \cap \beta]\!]\delta &= [\![\alpha]\!]\delta \cap [\![\beta]\!]\delta & [\![\alpha \cup \beta]\!]\delta &= [\![\alpha]\!]\delta \cup [\![\beta]\!]\delta \\
[\![\sqcap_{d:D} \alpha]\!]\delta &= \bigcap_{v \in \mathbb{D}} [\![\alpha]\!]\delta[v/d] & [\![\sqcup_{d:D} \alpha]\!]\delta &= \bigcup_{v \in \mathbb{D}} [\![\alpha]\!]\delta[v/d]
\end{aligned}$$

Definition 2.8. A formula ϕ is a formula in the *modal μ-calculus* iff it is generated by the following grammar:

$$\begin{aligned}
\phi, \psi ::= \; & b \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall d{:}D.\, \phi \mid \exists d{:}D.\, \phi \mid \\
& [\alpha]\phi \mid \langle\alpha\rangle\phi \mid \mu X.\, \phi \mid \nu X.\, \phi \mid X
\end{aligned}$$

Here, b is a term of type B, α is an action formula and $X \in \mathcal{F}$ is a fixpoint variable, taken from some countable set $\mathcal{F}$. Every fixpoint variable represents a set of states; $\mu X.\, \phi$ and $\nu X.\, \phi$ are the least and greatest fixpoints over X respectively.

Since we do not need fixpoints with parameters in the examples, our definition excludes fixpoint parameters. This is a slight deviation from the standard definition of the first-order μ-calculus.

On top of the operators defined above, we also use the implication operator $\phi \Rightarrow \psi$ as an abbreviation of $\neg\phi \vee \psi$. The binding of the modal operators $[\alpha]$ and $\langle\alpha\rangle$ takes precedence over the binding of logical operators, *i.e.*, $[\alpha]\phi \wedge \psi$ should be interpreted as $([\alpha]\phi) \wedge \psi$.

To ensure the semantics is well-defined and for the purpose of model checking, we place several practical restrictions on the shape of μ-calculus formulae. Firstly, we say the occurrence of a data variable d (resp. fixpoint variable X) is *bound* iff it occurs within the scope of a formula $\mathsf{Q}d{:}D.\, \phi$, with $\mathsf{Q} \in \{\forall, \exists\}$ (resp. $\sigma X.\, \phi$, with $\sigma \in \{\mu, \nu\}$). Otherwise, the occurrence is *free*; the set of all data variables that occur freely in ϕ is denoted $\mathsf{vars}(\phi)$. A μ-calculus formula ϕ is *closed* iff no data

variables and no fixpoint variables occur freely in ϕ. We say a μ-calculus formula is *well-named* iff data variables and fixpoint variables are not bound more than once, *i.e.*, the variable d (resp. X) in the binding $\mathsf{Q}d{:}D.\,\phi$ (resp. $\sigma X.\,\phi$) is not bound anywhere else. Finally, a μ-calculus formula is *monotone* iff the number of negations (including those arising from implications) that occurs between every fixpoint variable X and its binding $\sigma X.\,\phi$ is even. From here on, we only consider μ-calculus formulae that are well-named and monotone. Furthermore, the formulae we consider in the examples are also closed. Any monotone μ-calculus formula can be transformed into an equivalent *normalised* formula, in which only Boolean expressions b may occur in the scope of a negation.

Semantics Formulae in the μ-calculus are interpreted over a labelled transition system $(S, \to, \hat{s})$ and can be evaluated in each state $s \in S$. We first give the intuition behind the semantics; a formal definition follows. The box modality $[a]\phi$ expresses that after every a-transition starting from s, ϕ must hold. If no a-transition is enabled in s, then $[a]\phi$ vacuously holds in s. The diamond modality $\langle a\rangle\phi$ is *true* if and only if an a-transition is possible in s after which ϕ holds. The least fixpoint $\mu X.\,\phi$ is true for the smallest set of states X such that ϕ holds for all states in X (hence the name 'least fixpoint'). Dually, $\nu X.\,\phi$ is true for the largest set X that satisfies ϕ. In a typical formula with fixpoints, X occurs in the body ϕ of its binding $\sigma X.\,\phi$ and in the scope of a modal operator. An example is $\nu X.\langle a\rangle X$, "there is an infinite path consisting only of a actions". In this way, the greatest fixpoint operator is used to express infinite behaviour, while the least fixpoint operator is can express finite behaviour.

These ideas are formalised below. In this definition, we use a *fixpoint environment* $\eta : \mathcal{F} \to 2^S$ to interpret the semantics of fixpoint variables.

Definition 2.9. Given an LTS $(S, \to, \hat{s})$, the semantics of a μ-calculus formula ϕ in the context of a fixpoint environment η and a data environment δ is $[\![\phi]\!]\eta\delta \subseteq S$, defined inductively as follows:

$$[\![b]\!]\eta\delta = \begin{cases} S & \text{if } [\![b]\!]\delta \\ \emptyset & \text{otherwise} \end{cases} \qquad [\![\neg\phi]\!]\eta\delta = S \setminus [\![\phi]\!]\eta\delta$$

$$[\![\phi \wedge \psi]\!]\eta\delta = [\![\phi]\!]\eta\delta \cap [\![\psi]\!]\eta\delta \qquad [\![\phi \vee \psi]\!]\eta\delta = [\![\phi]\!]\eta\delta \cup [\![\psi]\!]\eta\delta$$

$$[\![\forall d{:}D.\,\phi]\!]\eta\delta = \bigcap_{v \in \mathbb{D}} [\![\phi]\!]\eta\delta[v/d] \qquad [\![\exists d{:}D.\,\phi]\!]\eta\delta = \bigcup_{v \in \mathbb{D}} [\![\phi]\!]\eta\delta[v/d]$$

$$[\![[\alpha]\phi]\!]\eta\delta = \{s \mid \forall a \in [\![\alpha]\!]\delta, t \in S.\, s \xrightarrow{a} t \Rightarrow t \in [\![\phi]\!]\eta\delta\}$$

$$[\![\langle\alpha\rangle\phi]\!]\eta\delta = \{s \mid \exists a \in [\![\alpha]\!]\delta, t \in S.\, s \xrightarrow{a} t \wedge t \in [\![\phi]\!]\eta\delta\}$$

$$[\![\mu X.\,\phi]\!]\eta\delta = \bigcap\{T \subseteq S \mid [\![\phi]\!]\eta[T/X]\delta \subseteq T\}$$

$$[\![\nu X.\,\phi]\!]\eta\delta = \bigcup\{T \subseteq S \mid T \subseteq [\![\phi]\!]\eta[T/X]\delta\}$$

$$[\![X]\!]\eta\delta = \eta(X)$$

Remark that $[\![\phi]\!]\eta\delta$ does not depend on η and δ if ϕ is a closed formula. In those cases, we will also write $[\![\phi]\!]$. An LTS $TS = (S, \rightarrow, \hat{s})$ satisfies a closed formula ϕ, notation $TS \models \phi$, iff $\hat{s} \in [\![\phi]\!]$. By extension, an LPS satisfies a formula ϕ iff its LTS satisfies ϕ.

The existence of the smallest and largest fixpoints in the complete lattice $(2^S, \subseteq)$ follows from the monotonicity of ϕ and the Knaster-Tarski theorem [78, 123]. Note the duality between $\langle\alpha\rangle\phi$ and $[\alpha]\phi$ and also between $\mu X.\,\phi$ and $\nu X.\,\phi$.

Example 2.10. Recall again the LTS from Example 2.3. We want to check several properties on this LTS.

Firstly, to check for deadlocks, we use the formula

$$\nu X.([\top]X \wedge \langle\top\rangle \mathit{true}) \tag{2.1}$$

which expresses "at every point in every trace, there exists some action after which *true* holds". This is equivalent to stating that in any reachable state, at least one action is enabled. This formula does not hold for the LTS of the ATM, since the state reached after performing *empty* is a deadlock.

Secondly, consider the property "after the ATM is empty, it will never allow inserting the card again". An equivalent wording is "after any sequence of actions that ends with *empty*, there is no finite trace that ends with the action *insert_card*", which can be formalised as

$$\nu X.\Big([\top]X \wedge [\mathit{empty}]\neg\mu Y.\,(\langle\top\rangle Y \vee \langle \mathit{insert\_card}\rangle \mathit{true})\Big)$$

Remark that this formula is monotone: the only negation does not occur between a fixpoint variable and its binding. The equivalent normalised formula is

$$\nu X.\Big([\top]X \wedge [\mathit{empty}]\nu Y.\,([\top]Y \wedge [\mathit{insert\_card}]\mathit{false})\Big) \tag{2.2}$$

The LTS of the ATM satisfies this property.

To check whether the ATM always ejects the desired amount of cash, we use the formula

$$\nu X.\,([\top]X \wedge \forall n, n'{:}N.\,[\mathit{amount}(n)][\mathit{cash}(n')](n = n')) \tag{2.3}$$

which requires that whenever two actions $\mathit{amount}(n)$ and $\mathit{cash}(n')$ happen successively, their parameters n and n' have the same value. This is indeed the case, so the formula holds.

Lastly, we have the property "after inserting a card, and not entering the PIN incorrectly infinitely often, cash will unavoidably be ejected". Rephrasing this requirement in terms of actions and traces yields: "after *insert_card* happens, all traces contain *wrong* infinitely often or perform *cash* after a finite number of steps, while no deadlock occurs in the meantime", which can be formalised as follows:

$$\begin{aligned}\nu X.\Big([\top]X \wedge [\mathit{insert\_card}] \\ \nu Y.\,\mu Z.\,\big([\mathit{wrong}]Y \wedge [\,\overline{\mathit{wrong} \cup \sqcup_{n:N}\ \mathit{cash}(n)}\,]Z \wedge \langle\top\rangle \mathit{true}\big)\Big)\end{aligned} \tag{2.4}$$

This property is also satisfied by the ATM LTS. □

The complexity of a μ-calculus formula can be formalised by the concept of *alternation depth*: the more alternations between least fixpoints μ and greatest fixpoints ν a formula contains, the harder it is to check and, incidentally, to understand. Formally, the alternation depth of a fixpoint variable X within a normalised formula ϕ, notation $AltDepth_\phi(X)$, is the length of the longest sequence $\sigma_1 X_1.\,\phi_1 \ldots \sigma_n X_n.\,\phi_n$ of subformulae of ϕ such that $X = X_1$ and, for all $1 < i \leq n$, we have $\sigma_i \neq \sigma_{i-1}$ and X_{i-1} occurs freely in $\sigma_i X_i.\,\phi_i$. The alternation depth of a normalised formula ϕ is the maximum alternation depth of the fixpoint variables that occur in it, or 0 if there are none. In Example 2.10, the first three formulas have an alternation depth of one and the last formula has an alternation depth of two, due to variable Y. Most real-world formulas have a small alternation depth, typically not larger than three.

Other Temporal Logics Other popular logics for verification include *linear temporal logic* (LTL) [116], *computation tree logic* (CTL) [30] and *extended computation tree logic* (CTL*) [41]. We will not formalise them here, but informally describe which fragment of the μ-calculus they represent.

Firstly, as the name indicates, LTL only considers linear behaviour, *i.e.*, single paths. The semantics is such that an LTL formula holds in a state iff it holds for all paths in that state. Furthermore, the semantics limits the number of meaningful fixpoint alternations. Thus, LTL roughly corresponds to the μ-calculus without the diamond operator $\langle\alpha\rangle$ and with a maximum alternation depth of two. Secondly, CTL *does* consider branching behaviour, but further limits fixpoint alternations. Hence, the expressive power of CTL is roughly similar to the μ-calculus without fixpoint alternations.

Since LTL and CTL are incomparable [41], their combination, CTL*, is strictly more expressive than either logic. However, the μ-calculus is even more expressive: every CTL* formula can be translated into an equivalent μ-calculus formula with a worst-case exponential increase in the size of the formula [18] (or a linear increase if first-order constructions are allowed [35]). An important practical difference between μ-calculus and LTL/CTL is that μ-calculus is often applied in the setting of action-based semantics, *i.e.*, an LTS. LTL and CTL, on the other hand, are typically used in the verification of models with state-based semantics, where the states are labelled instead of the edges.

2.4 Parity Games

In this section, we introduce *parity games* [43, 97] and the related concepts. Parity games can encode various decision problems, among them μ-calculus model checking of LTSs (see Section 2.4.1). A parity game is played by two players, *even* ($\Diamond$) and *odd* ($\Box$), on a directed graph.

Definition 2.11. A *parity game* is a directed graph $G = (V, E, \Omega, \mathcal{P})$, where

- V is a set of nodes, called the *state space*;
- $E \subseteq V \times V$ is a transition relation;
- $\Omega : V \to \mathbb{N}$ is a function that assigns a *priority* to each node; and
- $\mathcal{P} : V \to \{\Diamond, \Box\}$ is a function that assigns an *owner* to each node.

Our definition deviates slightly from the standard formalisation: we use the function $\mathcal{P}$ to define the owner of each node, instead of two sets $V_\Diamond$ and $V_\Box$ that partition V. We write $s \to t$ whenever $(s,t) \in E$. The set of successors of a node s is denoted with $succ(s) = \{t \mid s \to t\}$. We use $\bigcirc$ to denote an arbitrary player and $\bar{\bigcirc}$ to denote its opponent. In the parity games we consider, the set of priorities assigned by Ω is bounded.

A parity game is played as follows: initially, a token is placed on some node of the graph. Next, the owner of the node can decide where to move the token; the token must be moved along one of the outgoing edges and ends up in the corresponding target node. This either continues ad infinitum or until the token gets stuck in a node that does not have any outgoing edges. The resulting maximal sequence of nodes that the token has moved through is called a *play*. Finite plays are *won* by whichever player does *not* own the last node on the play. An infinite play is won by player $\Diamond$ iff the minimal priority that occurs infinitely often along the play is even. Otherwise, it is won by player $\Box$. On every infinite play, there must be at least one priority that occurs infinitely often, due to boundedness of Ω.

Remark 2.12. The parity games that we consider in this thesis are *min parity games*, meaning that smaller priorities dominate larger priorities. *Max parity games*, where the parity of the *largest* priority that occurs infinitely often determines the winner of a play, also occur frequently in the literature. The concepts of min and max parity games coincide when the priority function Ω is bounded [54]. □

To reason about moves that a player may want to take, we use the concept of *strategies*. A strategy $\sigma_\bigcirc : V^+ \to V$ for player $\bigcirc$ is a partial function that determines where $\bigcirc$ moves the token next, after the token has passed through a finite sequence of nodes that ends in a node owned by $\bigcirc$. More formally, for all sequences $s_1 \dots s_n$ such that $\mathcal{P}(s_n) = \bigcirc$, it holds that $\sigma_\bigcirc(s_1 \dots s_n) \in succ(s_n)$. If s_n belongs to $\bar{\bigcirc}$, $\sigma_\bigcirc(s_1 \dots s_n)$ is undefined. A play $s_1, s_2, \dots$ is *consistent* with a strategy σ if and only if $\sigma(s_1 \dots s_i) = s_{i+1}$ for all i such that $\sigma(s_1 \dots s_i)$ is defined. A player $\bigcirc$ wins a node s if and only if there is a strategy $\sigma_\bigcirc$ such that all plays that start in s and that are consistent with $\sigma_\bigcirc$ are won by player $\bigcirc$.

Example 2.13. Consider the parity game in Figure 2.2. Here, priorities are inscribed in the nodes and the nodes are shaped according to their owner ($\Diamond$ or $\Box$). Let π be an arbitrary, possibly empty, sequence of nodes. In this game, the strategy $\sigma_\Diamond$, partially defined as $\sigma_\Diamond(\pi s_1) = s_2$ and $\sigma_\Diamond(\pi s_2) = s_1$, is winning for $\Diamond$ in s_1 and s_2. After all, the minimal priority that occurs infinitely often along $(s_1 s_2)^\omega$ is 0, which is

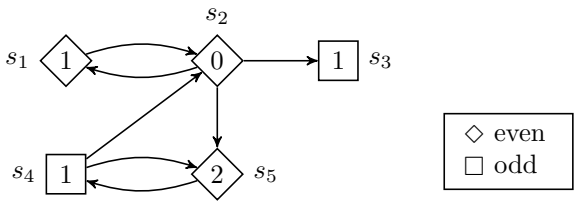


Figure 2.2: A parity game with five nodes, three belonging to player ◇ and two belonging to player □. Priorities are inscribed in the nodes.

even. Alternatively, player ◇ can choose to move the token from s_2 to s_3, at which point player □ gets stuck and loses.

Player □ can win node s_4 with the strategy $\sigma_\Box(\pi s_4) = s_5$. Note that player ◇ is always forced to move the token from node s_5 to s_4. □

In case a parity game has a total transition relation, in which every node has at least one outgoing edge, we call that game *total*. In the literature, parity games are commonly restricted to be total, since it means one only has to reason about infinite plays. Note that a parity game with a non-total transition relation can easily be transformed into a parity game with a total transition relation: by adding two nodes $\boxed{0}\circlearrowleft$ and $\boxed{1}\circlearrowleft$, and forcing all finite plays that are winning for player ◇, respectively □, to those nodes.

Dominions and Subgames Given a strategy σ for player ○, a set of nodes U is called a *σ-dominion* if and only if every play which starts in U and is consistent with σ, stays in U and is won by ○. Furthermore, U is called a *○-dominion* iff U is a σ-dominion for some ○-strategy σ. If U is a ○-dominion for some player ○, then we simply call it a *dominion*.

Given a game $G = (V, E, \Omega, \mathcal{P})$, we consider two types of *subgames*. Firstly, we have subgames where the set of nodes is restricted by a set $U \subseteq V$: $G \cap U$ denotes the subgame $(U, E \cap (U \times U), \Omega|_U, \mathcal{P}|_U)$, where $f|_A$ denotes the function f with its domain restricted to A. Note that $G \cap U$ is not necessarily total, even if G is total. However, when G is total and U is a dominion, then $G \cap U$ is also a total parity game.

Secondly, we consider subgames where the edge relation is restricted by a strategy σ. This is denoted $G \cap \sigma$, which is defined as $(V, E_\sigma, \Omega, \mathcal{P})$, where E_σ is such that, for each node $s \in V$, the successor set sE_σ of s is defined as

$$sE_\sigma = \begin{cases} \{\sigma(s)\} & \text{if } \sigma(s) \text{ is defined} \\ sE & \text{otherwise} \end{cases}$$

Remark that in $G \cap \sigma$, only one player can make non-trivial choices. Such a game is called *solitaire*.

Solving A nice property of parity games is that they are *determined* [43]: every node is won by one of the players. Moreover, they are also *positionally determined* [140], which means that every node is won by one of the players with a *memoryless strategy* $\sigma : V \to V$. A memoryless strategy directs the token based only on the current node, and does not take previous moves of the token into account. In most of the thesis we use memoryless strategies; in only a few proofs we require strategies with memory.

Solving a parity game means computing a partitioning $\{W_\Diamond, W_\Box\}$ of the nodes into those nodes won by player $\Diamond$ ($W_\Diamond$) and those won by player $\Box$ ($W_\Box$). A witness to a solution is a pair of strategies $(\sigma_\Diamond, \sigma_\Box)$ such that $W_\Diamond$ is a $\sigma_\Diamond$-dominion and $W_\Box$ is a $\sigma_\Box$-dominion. Parity game solving is one of the few problems that is in *UP* and in *co-UP* [68], but for which no polynomial time algorithm is known (yet). A recursive algorithm for solving parity games follows directly from Zielonka's proof of positional determinacy [140].

2.4.1 Application in Model Checking

One of the applications of parity games is model checking: given an LTS $TS = (S, \to, \hat{s})$ and a μ-calculus formula ϕ, one can construct a corresponding parity game which contains a node (s, δ, ψ) for every combination of state s, subformula ψ and data environment δ (the latter is not relevant if ϕ is closed). Since the node $(\hat{s}, \delta, \phi)$ is won by player $\Diamond$ if and only if $TS \vDash [\![\phi]\!]\delta$, solving the parity game also answers the model checking problem [24]. Below, Φ is the set of all μ-calculus formulae and $\phi[e/d]$ (resp. $\phi[\psi/X]$) denotes the substitution of d by e (resp. X by ψ) in ϕ. Recall that Δ is the set of all data environments.

Definition 2.14. Given an LTS $TS = (S, \to, \hat{s})$ and a closed and normalised μ-calculus formula ϕ, the corresponding parity game is $G = (S \times \Delta \times \Phi, E, \Omega, \mathcal{P})$, where E, Ω and $\mathcal{P}$ are defined in Table 2.1.

Remark that the recursion in Table 2.1 never arrives at a formula X, since we always replace those by their binding subformula. Furthermore, the nodes corresponding to a least fixpoint $\mu X.\, \phi$ are always assigned an odd priority and, $\nu X.\, \phi$-nodes are assigned an even priority. This is in accordance with the semantics of the μ-calculus, as per the following theorem.

Theorem 2.15 ([24])**.** *Let $TS = (S, \to, \hat{s})$ be an LTS and ϕ a closed and normalised μ-calculus formula. Then, $TS \models \phi$ iff player $\Diamond$ wins $(\hat{s}, \delta, \phi)$, where δ is arbitrary, in the corresponding parity game.*

We conclude this section by encoding our running example into a parity game.

Example 2.16. We revisit the LTS of an ATM from Example 2.3 and the properties specified in Example 2.10, specifically formula 2.4. A compacted representation of the corresponding parity game is displayed in Figure 2.3. Here, formulas of the shape $[a]\phi \wedge [b]\psi$ are represented by one node belonging to player $\Box$, instead of three nodes.

Table 2.1: Translation scheme of an LTS and a formula ϕ into a parity game. Below, N denotes the alternation depth of ϕ.

node	successors	Ω	$\mathcal{P}$
(s, δ, b)	$\emptyset$	N	$\begin{cases}\Box & \text{if } [\![b]\!]\delta \\ \Diamond & \text{if } \neg[\![b]\!]\delta\end{cases}$
$(s, \delta, \psi_1 \wedge \psi_2)$	$\{(s, \delta, \psi_1), (s, \delta, \psi_2)\}$	N	$\Box$
$(s, \delta, \psi_1 \vee \psi_2)$	$\{(s, \delta, \psi_1), (s, \delta, \psi_2)\}$	N	$\Diamond$
$(s, \delta, \forall d{:}D.\, \psi)$	$\{(s, \delta[v/d], \psi) \mid v \in \mathbb{D}\}$	N	$\Box$
$(s, \delta, \exists d{:}D.\, \psi)$	$\{(s, \delta[v/d], \psi) \mid v \in \mathbb{D}\}$	N	$\Diamond$
$(s, \delta, [\alpha]\psi)$	$\{(t, \delta, \psi) \mid a \in [\![\alpha]\!]\delta \wedge s \xrightarrow{a} t\}$	N	$\Box$
$(s, \delta, \langle\alpha\rangle\psi)$	$\{(t, \delta, \psi) \mid a \in [\![\alpha]\!]\delta \wedge s \xrightarrow{a} t\}$	N	$\Diamond$
$(s, \delta, \mu X.\, \psi)$	$\{(s, \delta, \psi[\mu X.\, \psi/X])\}$	$2\lfloor N - \mathit{AltDepth}_\phi(X)/2 \rfloor + 1$	$\Box$
$(s, \delta, \nu X.\, \psi)$	$\{(s, \delta, \psi[\nu X.\, \psi/X])\}$	$2\lfloor N - \mathit{AltDepth}_\phi(X)/2 \rfloor$	$\Box$

Furthermore, we omitted the nodes corresponding to the subformula $\langle\top\rangle \mathit{true}$, since they are not relevant for the solution of the parity game.

Remark that fixpoint X covers the complete state space of the LTS; fixpoints Y and Z only cover the part that is related to entering the PIN and receiving cash. All nodes in the game are won by player $\Diamond$, since all paths either end in a node owned by $\Box$ or the priority 0 occurs infinitely often on them. Thus, we again conclude that the ATM unavoidably delivers cash after entering the correct PIN. $\Box$

2.5 Parameterised Boolean Equations Systems

In the previous section, we saw how the model checking problem can be reduced to solving parity games by first generating an LTS and then constructing the corresponding parity game based on the μ-calculus formula. This section introduces *parameterised Boolean equations systems* (PBESs) [55, 59, 95], which can serve as a high-level representation of parity games.

Definition 2.17. A *predicate formula* is defined by the following grammar:

$$\phi, \psi ::= b \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall e{:}E.\, \phi \mid \exists e{:}E.\, \phi \mid X(f)$$

where b is a data term of sort B, e is a variable of sort E, X is a predicate variable of sort $D \to B$, which is taken from some set $\mathcal{X}$ of sorted predicate variables and argument f is an expression of sort D. The interpretation of a predicate formula ϕ in the context of a *predicate environment* $\eta : \mathcal{X} \to 2^{\mathbb{D}}$, providing an interpretation for predicate variables from $\mathcal{X}$, and a data environment δ is denoted by $[\![\phi]\!]\eta\delta$ and

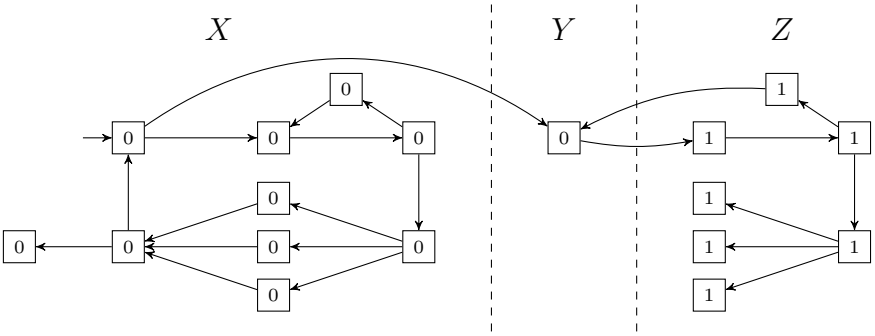


Figure 2.3: Parity game corresponding to checking formula 2.4 on the LTS of Figure 2.1. The dashed lines demarcate the nodes that originate from the fixpoints X, Y and Z.

inductively defined as follows:

$$\begin{aligned}
[\![b]\!]\eta\delta &\Leftrightarrow [\![b]\!]\delta \text{ holds} & [\![\neg\varphi]\!]\eta\delta &\Leftrightarrow [\![\varphi]\!]\eta\delta \text{ does not hold}\\
[\![\varphi \wedge \psi]\!]\eta\delta &\Leftrightarrow [\![\varphi]\!]\eta\delta \text{ and } [\![\psi]\!]\eta\delta \text{ hold} & [\![\varphi \vee \psi]\!]\eta\delta &\Leftrightarrow [\![\varphi]\!]\eta\delta \text{ or } [\![\psi]\!]\eta\delta \text{ holds}\\
[\![\forall d{:}E.\, \varphi]\!]\eta\delta &\Leftrightarrow \text{for all } v \in \mathbb{E},\ [\![\varphi]\!]\eta\delta[v/d] \text{ holds}\\
[\![\exists d{:}E.\, \varphi]\!]\eta\delta &\Leftrightarrow \text{for some } v \in \mathbb{E},\ [\![\varphi]\!]\eta\delta[v/d] \text{ holds}\\
[\![X(f)]\!]\eta\delta &\Leftrightarrow [\![f]\!]\delta \in \eta(X)
\end{aligned}$$

A couple of concepts for predicate formulae are defined analogously to those for μ-calculus formulae. Firstly, we again use $\phi \Rightarrow \psi$ as a shorthand for $\neg\phi \vee \psi$. A predicate formula is *monotone* iff all predicate variables occur in the scope of an even number of negations. Furthermore, we say a predicate formula is *normalised* iff negations only occur before a Boolean term b. Note that every monotone formula can be normalised by distributing negation over the other operators and eliminating double negations. Bound and free occurrence of data variables is defined similar as in the μ-calculus; $\mathsf{vars}(\varphi)$ again denotes the set of variables that occur freely in φ. A predicate formula is called *simple* iff no predicate variables occur in it.

Definition 2.18. A *parameterised Boolean equation system* (PBES) is a sequence of equations as defined by the following grammar:

$$\mathcal{E} ::= \emptyset \mid (\nu X(d{:}D) = \varphi)\mathcal{E} \mid (\mu X(d{:}D) = \varphi)\mathcal{E}$$

where $\emptyset$ is the empty PBES, μ and ν denote the least and greatest fixpoint operator, respectively, and $X \in \mathcal{X}$ is a predicate variable of sort $D \to B$. The right-hand side φ is a syntactically monotone predicate formula. Lastly, $d \in \mathcal{V}$ is a parameter of sort D.

As with LPSs, in the majority of the thesis, we only consider parameterised Boolean equation systems where each equation carries the same single parameter of a given

data sort D. This does not affect the generality of the theory we develop. The examples may contain equations with multiple parameters.

We use $\mathsf{bnd}(\mathcal{E})$ to denote the predicate variables bound in $\mathcal{E}$, *i.e.*, those variables occurring at the left-hand side of an equation. For an equation for X, d_X denotes its parameter and φ_X denotes its right-hand side predicate formula. We omit the trailing $\emptyset$. We say a PBES is *closed* iff it does not contain free variables, *i.e.*, all data variables that occur in a right-hand side φ_X are either bound by a quantifier or as a data parameter of X, whereas all predicate variables belong to $\mathsf{bnd}(\mathcal{E})$. A PBES $\mathcal{E}$ is called a *Boolean equation system* (BES) [92] iff all predicate variables bound by $\mathcal{E}$ have type $D_\star \to B$ and every right-hand side only contains the operators $\wedge$ and $\vee$, constants *true* and *false* and $X(\star)$. We say that a PBES $\mathcal{E}$ is *well-formed* iff for every $X \in \mathsf{bnd}(\mathcal{E})$ there is exactly one equation in $\mathcal{E}$. In the remainder of the thesis we only reason about well-formed, closed PBESs.

Definition 2.19. The *solution* $[\![\mathcal{E}]\!]\eta\delta$ of a PBES $\mathcal{E}$ in the context of a predicate environment η and a data environment δ, is a predicate environment that is defined inductively:

$$
\begin{aligned}
[\![\,\emptyset\,]\!]\eta\delta &= \eta \\
[\![(\mu X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta &= [\![\mathcal{E}]\!]\eta[\mu T_X/X]\delta \\
[\![(\nu X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta &= [\![\mathcal{E}]\!]\eta[\nu T_X/X]\delta
\end{aligned}
$$

with $T_X(R) = \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}]\!]\eta[R/X]\delta)\delta[v/d]\}$.

Intuitively, the solution of a PBES gives priority to fixpoints that occur early in the PBES, while satisfying the equalities that are specified by each equation. The monotonicity of the transformer $T_X \colon 2^{\mathbb{D}} \to 2^{\mathbb{D}}$, which follows from syntactic monotonicity of φ_X, guarantees the existence of the least fixpoint μT_X and greatest fixpoint νT_X in the complete lattice $(2^{\mathbb{D}}, \subseteq)$. Also, note that the solution of a bound variable in a closed PBES does not depend on the environments η and δ. For this reason, we often omit η and δ and simply write $[\![\mathcal{E}]\!]$ instead of $[\![\mathcal{E}]\!]\eta\delta$. Finally, for a PBES $\mathcal{E}$ and some $X \in \mathsf{bnd}(\mathcal{E})$ we sometimes say that (the solution to) $X(v)$ is *true* iff $v \in [\![\mathcal{E}]\!](X)$.

Example 2.20. Consider the following PBES $\mathcal{E}$ consisting of an equation for X and an equation for Y, both carrying a single parameter. Furthermore, the equation for X has a least fixpoint, and the equation for Y has a greatest fixpoint.

$$
\begin{aligned}
\mu X(n{:}N) &= (\exists m{:}N.\, m \geq n \wedge X(m)) \wedge Y(\mathit{false}) \\
\nu Y(b{:}B) &= Y(\neg b)
\end{aligned}
$$

By applying the semantics of predicate formulae, we can derive the predicate trans-

former for Y as follows:

$$\begin{aligned} T_Y(R) &= \{v \in \mathbb{B} \mid [\![Y(\neg b)]\!]([\![\emptyset]\!]\eta[R/Y]\delta)\delta[v/b]\} \\ &= \{v \in \mathbb{B} \mid [\![Y(\neg b)]\!]\eta[R/Y]\delta[v/b]\} \\ &= \{v \in \mathbb{B} \mid [\![\neg b]\!]\delta[v/b] \in \eta[R/Y](Y)\} \\ &= \{v \in \mathbb{B} \mid \neg v \in R\} \end{aligned}$$

The largest set that satisfies $T_Y(R) = R$ is $\mathbb{B}$, hence $\nu T_Y = \mathbb{B}$. We can apply a similar reasoning to X to obtain its predicate transformer.

$$\begin{aligned} T_X(R) = \{v \in \mathbb{N} \mid [\![&(\exists m{:}N.\, m \geq n \wedge X(m)) \wedge Y(\mathit{false})]\!] \\ &([\![\nu Y(b{:}B) = Y(\neg b)]\!]\eta[R/X]\delta)\delta[v/n]\} \\ = \{v \in \mathbb{N} \mid \exists v' \in \mathbb{N}.\, v' \geq v \wedge v' \in R\} \end{aligned}$$

We derive that $\mu T_X = \emptyset$. The application of Definition 2.19 yields $[\![\mathcal{E}]\!]\eta\delta = \eta[\mu T_X/X][\nu T_Y/Y]$. The solution of $\mathcal{E}$ thus satisfies $[\![\mathcal{E}]\!](X) = \emptyset$ and $[\![\mathcal{E}]\!](Y) = \mathbb{B}$. Note that since this particular example is not mutually recursive, the order of the equations does not influence the solution. □

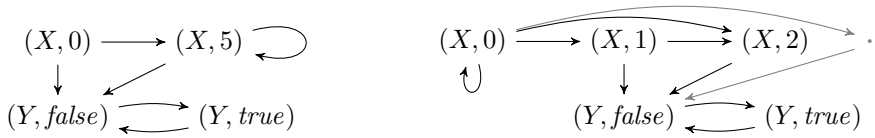
2.5.1 Dependency Graphs and Proof Graphs

Although the semantics for PBESs given in Definition 2.19 is sufficient to compute the solution and reason about the correctness of PBES transformations, its denotational nature is not very intuitive. This is also demonstrated by Example 2.20, which required relatively complex reasoning, even for a small PBES. To provide an operational view on PBESs and their solution, Cranen *et al.* developed the notion of *dependency graphs* [37]. Before we introduce these graphs formally, we need some additional concepts.

First, $\mathsf{sig}(\mathcal{E})$ is the signature of $\mathcal{E}$, defined as $\mathsf{sig}(\mathcal{E}) = \{(X, v) \mid X \in \mathsf{bnd}(\mathcal{E}), v \in \mathbb{D}\}$. For a given set $S \subseteq \mathsf{sig}(\mathcal{E})$, the predicate environment $\mathsf{env}(S, \mathit{true})$ that follows from it is defined as $\mathsf{env}(S, \mathit{true})(X) = \{v \in \mathbb{D} \mid (X, v) \in S\}$. Dually, we define $\mathsf{env}(S, \mathit{false})(X) = \mathbb{D} \setminus \mathsf{env}(S, \mathit{true})(X)$. Furthermore, every predicate variable bound in $\mathcal{E}$ is assigned a *rank*, where $\mathsf{rank}_{\mathcal{E}}(X) \leq \mathsf{rank}_{\mathcal{E}}(Y)$ if X occurs before Y in $\mathcal{E}$, and $\mathsf{rank}_{\mathcal{E}}(X)$ is even if and only if X is labelled with a greatest fixed point. We assume every PBES has a fixed rank function.

Definition 2.21. Let $\mathcal{E}$ be a PBES and $G = (V, E)$ be a directed graph, where $V \subseteq \mathsf{sig}(\mathcal{E})$. We say G is a *dependency graph* for $r \in \mathbb{B}$ iff for every $(X, v) \in V$ and for all δ, $[\![\varphi_X]\!]\eta(\delta[v/d_X]) = r$ with $\eta = \mathsf{env}((X, v)E, r)$, where sE denotes the successor set of a node, defined as $sE = \{t \mid s\, E\, t\}$.

We distinguish *positive dependency graphs*, where r is *true*, from *negative dependency graphs*, where r is *false*. Intuitively, in a positive dependency graph, $\eta = \mathsf{env}((X, v)E, \mathit{true})$ is a predicate environment that maps all successors of (X, v)



(a) A positive dependency graph **(b)** A negative dependency graph

Figure 2.4: Dependency graphs for the PBES from Example 2.20.

to *true* and all other nodes to *false*. Then, the requirement is that φ_X (and thus $X(v)$) is *true* under η and a data environment that maps d_X to v. In other words, the successors of a node (X, v) being *true* must imply that (X, v) is *true* as well. Dually, a negative dependency graph indicates a node (X, v) is *false*, when its successors are all *false*.

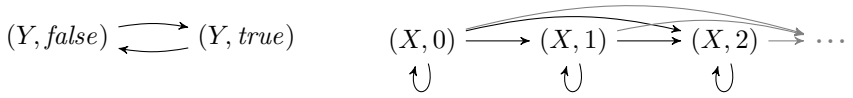
Example 2.22. Recall the PBES from Example 2.20. Figure 2.4 depicts a positive and a negative dependency graph for this PBES. We focus on node $(X, 0)$ in the positive dependency graph of Figure 2.4a. Its successors are $(X, 5)$ and (Y, false). The environment η induced by these successors is given by $\mathsf{env}((X, 0)E, \mathit{true})$, which sets these successors to *true*; *i.e.*, η is such that $\eta(X) = \{5\}$ and $\eta(Y) = \{\mathit{false}\}$. When we evaluate the right-hand side of the equation for X in the context of η and parameter n set to 0, we obtain $[\![(\exists m{:}N.\, m \geq n \wedge X(m)) \wedge Y(\mathit{false})]\!]\eta(\delta[0/n]) = \mathit{true}$. Therefore, the positive dependency graph condition is satisfied for node $(X, 0)$. A similar reasoning applies to the other nodes, showing that the dependency graph condition is satisfied by each of them.

Note that nodes (Y, false) and (Y, true) are dependent on each other in both dependency graphs. Furthermore, in the negative case (Figure 2.4b), $(X, 0)$ needs no dependency on (Y, false) as long as it depends on all (X, i) with $i \in \mathbb{N}$. Hence, this particular dependency graph is infinite. A finite negative dependency graph for $(X, 0)$ is $(X, 0) \longrightarrow (Y, \mathit{false}) \rightleftarrows (Y, \mathit{true})$. □

A dependency graph captures the logical structure of a PBES; it does not include the fixpoint semantics. If we want to reason about the actual solution of a PBES, we need an additional restriction on the infinite paths in a dependency graph. Dependency graphs that meet these restrictions are called *proof graphs*.

Definition 2.23. Let $G = (V, E)$ be a positive (respectively negative) dependency graph for a PBES $\mathcal{E}$. Then G is a *positive proof graph* (respectively *negative proof graph*) iff for all infinite paths π in G, the number $\min\{\mathsf{rank}_{\mathcal{E}}(X) \mid X \in V^{\infty}(\pi)\}$ is even (respectively odd), where $V^{\infty}(\pi)$ is the set of predicate variables that occur infinitely often along π.

Observe that predicate variables with a lower rank dominate those with a higher rank. This reflects the fact that fixpoint symbols that occur early in an equation system take priority over later ones (cf. Definition 2.19).



(a) A positive proof graph **(b)** A negative proof graph

Figure 2.5: Proof graphs for the PBES from Example 2.20.

Example 2.24. Recall again the PBES from Example 2.20. In this PBES, the rank of X is 1, and the rank of Y is 2. Figure 2.5 depicts a positive and a negative proof graph for this PBES. Note that Figure 2.5a depicts the smallest positive proof graph proving that $Y(\mathit{false})$ is *true*. Larger proof graphs can be obtained by adding a self loop to (Y, false) or (Y, true). Similarly, the proof graph in Figure 2.5b is the smallest negative proof graph explaining that $X(0)$ is *false*. However, there is a subgraph which shows that $X(1) = \mathit{false}$, *viz.* the graph that does not include $(X, 0)$. Note that for every $i \in \mathbb{N}$, the proof graph for (X, i) is infinite, since (X, i) depends on all (X, j) with $j \geq i$. □

The next theorem formally states the relationship between proof graphs and the solution of a PBES.

Theorem 2.25 ([37])**.** *Let $\mathcal{E}$ be a PBES with $X \in \mathsf{bnd}(\mathcal{E})$. Then $v \in [\![\mathcal{E}]\!](X)$ iff there is a positive proof graph (V, E) such that $(X, v) \in V$. Dually, $v \notin [\![\mathcal{E}]\!](X)$ iff there is a negative proof graph containing (X, v).*

2.5.2 Application in Model Checking

In Section 2.4.1, we saw that parity games can encode model checking problems on LTSs. Similarly, a PBES can encode the combination of an LPS and a μ-calculus formula [55, 58].

Definition 2.26. Let $L = (P, \hat{d})$ be an LPS and φ a closed and normalised μ-calculus formula, where P is defined as

$$P(d{:}D) = \sum_{i \in I} \sum_{e_i : E_i} \big(c_i(d, e_i) \to a_i(f_i(d, e_i)) \cdot P(g_i(d, e_i))\big)$$

Furthermore, let $(\sigma_1 X_1 . \varphi_1) \ldots (\sigma_n X_n . \varphi_n)$ be the sequence of fixpoint binding subformulae of φ such that $i < j$ iff $\sigma_i X_i . \varphi_i$ occurs before $\sigma_j X_j . \varphi_j$ in φ. Then, the corresponding PBES is

$$\begin{aligned} \nu X(d{:}D) &= \mathsf{rhs}(\varphi) \\ \sigma_1 X_1(d{:}D, \mathsf{vars}(\varphi_1)) &= \mathsf{rhs}(\varphi_1) \\ &\vdots \\ \sigma_n X_n(d{:}D, \mathsf{vars}(\varphi_n)) &= \mathsf{rhs}(\varphi_n) \end{aligned}$$

where the function rhs constructs the right-hand side of an equations as follows:

$$
\begin{aligned}
\mathsf{rhs}(b) &= b \\
\mathsf{rhs}(\psi_1 \wedge \psi_2) &= \mathsf{rhs}(\psi_1) \wedge \mathsf{rhs}(\psi_2) \\
\mathsf{rhs}(\psi_1 \vee \psi_2) &= \mathsf{rhs}(\psi_1) \vee \mathsf{rhs}(\psi_2) \\
\mathsf{rhs}(\forall e{:}E.\, \psi) &= \forall e{:}E.\, \mathsf{rhs}(\psi) \\
\mathsf{rhs}(\exists e{:}E.\, \psi) &= \exists e{:}E.\, \mathsf{rhs}(\psi) \\
\mathsf{rhs}([\alpha]\psi) &= \bigwedge_{i \in I} \forall e_i{:}E_i.\, (c_i(d, e_i) \wedge a_i(f_i(d, e_i)) \vdash \alpha) \Rightarrow \mathsf{rhs}(\psi)[g_i(d, e_i)/d] \\
\mathsf{rhs}(\langle\alpha\rangle\psi) &= \bigvee_{i \in I} \exists e_i{:}E_i.\, c_i(d, e_i) \wedge a_i(f_i(d, e_i)) \vdash \alpha \wedge \mathsf{rhs}(\psi)[g_i(d, e_i)/d] \\
\mathsf{rhs}(X_i) &= X_i(d, \mathsf{vars}(\varphi_i)) \\
\mathsf{rhs}(\mu X_i.\, \psi) &= X_i(d, \mathsf{vars}(\varphi_i)) \\
\mathsf{rhs}(\nu X_i.\, \psi) &= X_i(d, \mathsf{vars}(\varphi_i))
\end{aligned}
$$

and where $a(f) \vdash \alpha$ is a predicate formula that represents whether $a(f) \in [\![\alpha]\!]$, inductively defined as:

$$
\begin{aligned}
a(f) \vdash a'(f') &= \begin{cases} f = f' & \text{if } a = a' \\ \mathit{false} & \text{otherwise} \end{cases} & a(f) \vdash \overline{\alpha} &= \neg(a \vdash \alpha) \\
a(f) \vdash \bot &= \mathit{false} & a(f) \vdash \top &= \mathit{true} \\
a(f) \vdash \alpha \cap \beta &= a(f) \vdash \alpha \wedge a(f) \vdash \beta & a(f) \vdash \alpha \cup \beta &= a(f) \vdash \alpha \vee a(f) \vdash \beta \\
a(f) \vdash \sqcap_{d:D} \alpha &= \forall d{:}D.\, (a(f) \vdash \alpha) & a(f) \vdash \sqcup_{d:D} \alpha &= \exists d{:}D.\, (a(f) \vdash \alpha)
\end{aligned}
$$

In the resulting PBES, the equation for X is a "dummy fixpoint", meant to capture the satisfaction of φ, in case it does not start with a fixpoint. Data variables bound in quantifiers may occur freely in a subformula $\sigma X.\, \varphi$, and need to be captured as a parameter in the corresponding equation.

In practice, the predicate formulae emanating from the modal operators $[\alpha]$ and $\langle\alpha\rangle$ can become very large. Therefore, they are often simplified during construction: if the formula $a_i(f_i(d, e_i)) \vdash \alpha$ is unsatisfiable (which can often be decided based on action labels alone), then the whole clause corresponding to a_i is removed. The model checking PBESs we give in the examples are constructed in this way.

We have the following theorem on the correspondence between μ-calculus model checking of LPSs and solving PBESs:

Theorem 2.27 ([55])**.** *Let $L = (P, \hat{d})$ be an LPS, φ a closed and normalised μ-calculus formula and $\mathcal{E}$ be the corresponding PBES. Then, $L \vDash \varphi$ if and only if $[\![\hat{d}]\!] \in [\![\mathcal{E}]\!](X)$.*

We revisit the running example one last time and apply the above theorem.

Example 2.28. We construct the PBES that corresponds to the ATM LPS given in Example 2.6 and formula 2.4 from Example 2.10. We distinguish five subformulae that are significant for the predicate formulae contained in the PBES; they are indicated below.

$$\nu X.\Big(\overbrace{[\top]X}^{\phi_1} \wedge \underbrace{[\mathit{insert\_card}]\nu Y.\,\mu Z.\,\big(\overbrace{[\mathit{wrong}]Y}^{\phi_3} \wedge \overbrace{[\,\overline{\mathit{wrong} \cup \sqcup_{n:N}\ \mathit{cash}(n)}\,]Z}^{\phi_4} \wedge \overbrace{\langle\top\rangle\mathit{true}}^{\phi_5}\big)}_{\phi_2}\Big)$$

The corresponding PBES is:

$$\begin{aligned}
&\nu X(s{:}\mathit{State}, n{:}N) = \\
&\quad\Big(\begin{aligned}
&(s = \mathit{idle}) \Rightarrow X(\mathit{await_pin}, n) \\
&\wedge (s = \mathit{await_pin}) \Rightarrow X(\mathit{check_pin}, n) \\
&\wedge \forall b{:}B.\,(s = \mathit{check_pin}) \Rightarrow X(\mathit{if}(b, \mathit{await_amount}, \mathit{wrong_pin}), n) \\
&\wedge (s = \mathit{wrong_pin}) \Rightarrow X(\mathit{await_pin}, n) \\
&\wedge \forall m{:}N.\,(s = \mathit{await_amount} \wedge (m = 50 \vee m = 100 \vee m = 200)) \\
&\qquad\Rightarrow X(\mathit{deliver_cash}, m) \\
&\wedge (s = \mathit{deliver_cash}) \Rightarrow X(\mathit{check_cash}, 0) \\
&\wedge (s = \mathit{check_cash}) \Rightarrow X(\mathit{no_cash}, n) \\
&\wedge (s = \mathit{check_cash}) \Rightarrow X(\mathit{idle}, n)
\end{aligned}\Big\}\ \phi_1 \\
&\quad\wedge (s = \mathit{idle}) \Rightarrow Y(\mathit{await_pin}, n) \qquad\}\ \phi_2 \\
&\nu Y(s{:}\mathit{State}, n{:}N) = \\
&\quad Z(s, n) \\
&\mu Z(s{:}\mathit{State}, n{:}N) = \\
&\quad\Big(\begin{aligned}
&(s = \mathit{wrong_pin}) \Rightarrow Y(\mathit{await_pin}, n)
\end{aligned}\Big\}\ \phi_3 \\
&\quad\Big(\begin{aligned}
&(s = \mathit{idle}) \Rightarrow Z(\mathit{await_pin}, n) \\
&\wedge (s = \mathit{await_pin}) \Rightarrow Z(\mathit{check_pin}, n) \\
&\wedge \forall b{:}B.\,(s = \mathit{check_pin}) \Rightarrow Z(\mathit{if}(b, \mathit{await_amount}, \mathit{wrong_pin}), n) \\
&\wedge \forall m{:}N.\,(s = \mathit{await_amount} \wedge (m = 50 \vee m = 100 \vee m = 200)) \\
&\qquad\Rightarrow Z(\mathit{deliver_cash}, m) \\
&\wedge (s = \mathit{check_cash}) \Rightarrow Z(\mathit{no_cash}, n) \\
&\wedge (s = \mathit{check_cash}) \Rightarrow Z(\mathit{idle}, n)
\end{aligned}\Big\}\ \phi_4 \\
&\quad\Big(\begin{aligned}
&\wedge (s = \mathit{idle} \vee s = \mathit{await_pin} \vee (\exists b{:}B.\,s = \mathit{check_pin}) \vee s = \mathit{wrong_pin} \\
&\qquad\vee (\exists m{:}N.\,s = \mathit{await_amount} \wedge (m = 50 \vee m = 100 \vee m = 200)) \\
&\qquad\vee s = \mathit{deliver_cash} \vee s = \mathit{check_cash} \vee s = \mathit{check_cash} \vee s = \mathit{idle})
\end{aligned}\Big\}\ \phi_5
\end{aligned}$$

The predicate formulae that correspond to each of the μ-calculus subformulae are indicated on the right. The right-hand side of Y only consists of $Z(s, n)$. This is in accordance with the μ-calculus formula, where the fixpoint of Z is a direct subformula of the fixpoint Y. In the subformula that corresponds to ϕ_5, the condition $s = \mathit{check\_cash}$ occurs twice, since there are two actions which are guarded by this condition in the LPS. The solution of $\mathcal{E}$ is such that $(\mathit{idle}, 0) \in [\![\mathcal{E}]\!](X)$, and we again conclude that the ATM satisfies the property that cash is unavoidable ejected if one enters the PIN correctly. □

The relationships between the concepts we have discussed so far are summarised in Figure 2.6. The exact relation between PBESs and parity games will be studied in the next chapter.

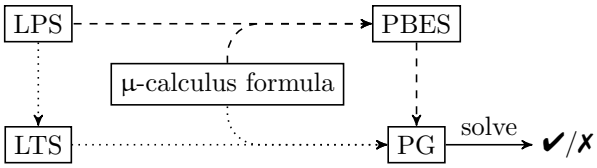
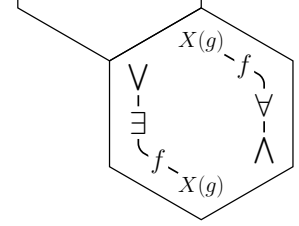


Figure 2.6: Model checking with PBESs (dashed arrows) as alternative for model checking with LTSs (dotted arrows). Vertical arrows correspond to state space exploration or instantiation, and may be impacted by the state explosion problem.

Chapter 3



Normal Forms for PBESs

The previous chapter introduced dependency graphs and proof graphs, which provide an operational explanation for the solution of a PBES. However, given the fact that a PBES can induce many different dependency graphs, it is hard to deduce what exactly the dependencies of a node (X, v) are. This is caused by the arbitrary nesting of operators that can occur in the right-hand side φ_X, meaning that many different, possibly overlapping successor sets $(X, v)E$ may satisfy the dependency condition in (X, v). Furthermore, a dependency graph for the PBES $(\nu X = X)(\nu Y = Y)$ may contain the the edge XEY, which is not related to any syntactic element of the PBES. Finally, each dependency graph captures only the positive or only the negative dependencies of a PBES. Knowledge about dependencies, both positive and negative, is a critical element of many techniques that aim to reduce the size of the underlying graph.

To address these issues, this chapter introduces two new normal forms for PBESs, called *standard recursive form* (SRF, Section 3.1) and *clustered recursive form* (CRF, Section 3.4). These normal forms facilitate the construction of a predicate formula that characterises whether a node (X, v) depends on some other node (Y, w). Using this, SRF allows us to construct a special kind of dependency graph, called *dependency space* (Section 3.2), which exactly describes positive and negative dependencies (Theorem 3.7). In that way, the new normal forms provide a framework that simplifies

reasoning about dependencies. Since dependency spaces coincide with parity games (Section 3.3), we can use existing solving techniques to find a proof graph within the dependency space and solve the PBES. Our normal forms generalise existing normal forms for BESs and improve over other normal forms for PBESs (Section 3.5). Whereas earlier normal forms may cause a quadratic blow-up in the size of the resulting PBES, the overhead of the SRF and CRF normal forms is limited: the number of new equations they introduce is bounded by the total size of all right-hand sides in the original PBES.

3.1 Standard Recursive Form

Standard recursive form for PBESs requires every right-hand side to be either disjunctive or conjunctive, as formalised below.

Definition 3.1. Let $\mathcal{E}$ be a PBES. Then $\mathcal{E}$ is in *standard recursive form* (SRF) iff for all $(\sigma_i X_i(d{:}D) = \phi) \in \mathcal{E}$, where $\sigma_i \in \{\mu, \nu\}$, ϕ is either disjunctive or conjunctive, *i.e.*, this equation for X_i has the shape

$$\sigma_i X_i(d{:}D) = \bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge X_j(g_j(d, e_j))$$

or

$$\sigma_i X_i(d{:}D) = \bigwedge_{j \in J_i} \forall e_j{:}E_j.\, f_j(d, e_j) \Rightarrow X_j(g_j(d, e_j))$$

We call each of the disjuncts or conjuncts of a right-hand side a *clause*. For a PBES $\mathcal{E}$ in SRF, we define a function $\mathsf{op}_{\mathcal{E}} : \mathsf{bnd}(\mathcal{E}) \to \{\wedge, \vee\}$ that indicates, for each predicate variable, whether its equation is conjunctive or disjunctive. We say a PBES in SRF is *total* if and only if for every $(X_i, v) \in \mathsf{sig}(\mathcal{E})$, at least one condition f_j should evaluate to *true*, *i.e.*, there is a $j \in J_i$ and a $v \in \mathbb{E}_j$ such that $[\![f_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j]$ holds. Henceforth, we assume that all PBESs in SRF are total.

We proceed to show how an arbitrary PBES can be transformed to a PBES in SRF. The intuition behind this transformation is as follows. Nested conjunctions, disjunctions and quantifiers are eliminated by introducing new predicate variables and extra equations for these variables. For instance, an equation $(\mu X(d{:}D) = \forall e{:}E.\, Y(d, e) \vee Z(d, e))$ can be replaced by two equations $(\mu X(d{:}D) = \forall e{:}E.\, \tilde{X}(d, e))$ $(\mu \tilde{X}(d{:}D, e{:}E) = Y(d, e) \vee Z(d, e))$ for some fresh variable $\tilde{X}$. Note that this results in at most a linear blow-up of the size of $\mathcal{E}$, since the number of new equations introduced is at most equal to the number of disjunctive/conjunctive alternations in all right-hand sides of $\mathcal{E}$. Furthermore, the number of new parameters introduced is bounded by the number of variables that occur in quantifiers.

The resulting SRF-PBES is made total by adding the equations $(\nu X_{\mathit{true}}(d{:}D_\star) = X_{\mathit{true}}(\star))$ and $(\mu X_{\mathit{false}}(d{:}D_\star) = X_{\mathit{false}}(\star))$ to $\mathcal{E}$, and adding a clause $\mathit{true} \Rightarrow X_{\mathit{true}}(\star)$ to every conjunctive right-hand side and a clause $\mathit{true} \wedge X_{\mathit{false}}(\star)$ to every disjunctive right-hand side (recall from Section 2.1 that $D_\star = \{\star\}$ is a singleton data sort).

Alternatively, the SRF-PBES can be made total by adding a clause $f_{neg} \Rightarrow X_{true}(\star)$ (resp. $f_{neg} \wedge X_{false}(\star)$), where f_{neg} is the complement of all other conditions. The condition f_{neg} can grow significantly, however, and effectively double the size of the right-hand side.

Before we formalise the SRF transformation, we first introduce a larger example, which we revisit several times in the rest of this chapter.

Example 3.2. Consider the PBES below.

$$\begin{aligned}\mu Y(n{:}N, b{:}B) &= \neg b \wedge (\neg(n > 0) \vee X(n-1)) \wedge (\neg(n > 5) \vee X(1))\\ \nu X(n{:}N) &= (\neg(0 < n \wedge n \leq 3) \vee Y(n, \mathit{true})) \wedge (X(0) \vee Y(7, \mathit{false}))\end{aligned}$$

Observe that the right-hand side of the equation for X contains an alternation between the operators $\wedge$ and $\vee$. The right-hand side of Y is purely conjunctive, but the expression $\neg b$ occurs without a predicate variable. An equivalent PBES in SRF is

$$\begin{aligned}\mu Y(n{:}N, b{:}B) &= (b \Rightarrow X_{\mathit{false}})\\ &\quad \wedge (n > 0 \Rightarrow X(n-1))\\ &\quad \wedge (n > 5 \Rightarrow X(1))\\ &\quad \wedge (\mathit{true} \Rightarrow X_{\mathit{true}})\\ \nu X(n{:}N) &= (0 < n \wedge n \leq 3 \Rightarrow Y(n, \mathit{true}))\\ &\quad \wedge (\mathit{true} \Rightarrow \tilde{X}(n))\\ \nu \tilde{X}(n{:}N) &= (\mathit{true} \wedge X(0))\\ &\quad \vee (\mathit{true} \wedge Y(7, \mathit{false}))\\ \mu X_{\mathit{false}} &= X_{\mathit{false}}\\ \nu X_{\mathit{true}} &= X_{\mathit{true}}\end{aligned}$$

The equation for $\tilde{X}$ has been introduced to eliminate the alternation in the right-hand side of X. The clause $\mathit{true} \Rightarrow X_{\mathit{true}}$ is necessary for $(Y, (0, \mathit{false}))$ to satisfy the restriction on totality. □

The transformation of arbitrary PBESs to SRF-PBESs is formalised by the rewriter R defined below. Here, we assume that the input PBES is normalised, *i.e.*, every right-hand side is a normalised predicate formula. The rewriter employs a number of auxiliary rewrite functions, $R_\vee$ and $R_\wedge$, that transform a normalised predicate formula to become disjunctive or conjunctive, respectively. Each of these rewriters returns a tuple with the transformed formula and a sequence of new equations that have to be added to the equation system.

In the following definitions, $V, W \subseteq \mathcal{V}$ are sets of typed variables, f and g are simple formulae and $\tilde{X} \in \mathcal{X}$ is a fresh predicate variable not occurring anywhere else in the PBES. We need to reason about the set of variables that are bound in a quantifier and, given a set $V = \{d_1, \ldots, d_n\}$, write $\exists V.\, \varphi$ instead of $\exists d_1, \ldots, d_n.\, \varphi$ (and similarly for $\forall W.\, \varphi$). To avoid name clashes, we assume that variables are never

bound more than once (as a parameter or in a quantifier). The rewriters $R_\vee$ and $R_\wedge$ carry a fixpoint symbol $\sigma \in \{\mu, \nu\}$ that is the fixpoint of the equation the current predicate formula appears in. The definition of $R_\vee$ is stated below. In four cases, we perform recursive calls on φ and/or ψ.

$$
\begin{aligned}
R_{\vee,\sigma}(f, V) &= \langle \exists d{:}D_\star.\, f \wedge X_{true}(\star), \emptyset \rangle \\
R_{\vee,\sigma}(X(e), V) &= \langle \exists d{:}D_\star.\, true \wedge X(e), \emptyset \rangle \\
R_{\vee,\sigma}(\varphi \wedge f, V) &= \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f \wedge f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\vee,\sigma}(\varphi, V) \\
R_{\vee,\sigma}(f \wedge \varphi, V) &= \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f \wedge f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\vee,\sigma}(\varphi, V) \\
R_{\vee,\sigma}(\varphi \wedge \psi, V) &= \langle \exists d{:}D_\star.\, true \wedge \tilde{X}(V), (\sigma \tilde{X}(V) = \varphi \wedge \psi) \rangle \\
&\qquad \text{where } \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\vee,\sigma}(\varphi, V) \\
&\qquad\quad \langle \textstyle\bigvee_{i \in I^\psi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\psi \rangle = R_{\vee,\sigma}(\psi, V) \\
R_{\vee,\sigma}(\varphi \vee \psi, V) &= \langle \textstyle\bigvee_{i \in I^\varphi \cup I^\psi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \mathcal{E}^\psi \rangle \\
R_{\vee,\sigma}(\forall W.\, f, V) &= \langle \exists d{:}D_\star.\, (\forall W.\, f) \wedge X_{true}(\star), \emptyset \rangle \\
R_{\vee,\sigma}(\forall W.\, \varphi, V) &= \langle \exists d{:}D_\star.\, true \wedge \tilde{X}(V), (\sigma \tilde{X}(V) = \forall W.\, \varphi) \rangle \\
R_{\vee,\sigma}(\exists W.\, \varphi, V) &= \langle \textstyle\bigvee_{i \in I^\varphi} \exists W \cup W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigvee_{i \in I^\varphi} \exists W_i.\, f_i \wedge X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\vee,\sigma}(\varphi, V \cup W)
\end{aligned}
$$

$R_\wedge$ is defined dually to $R_\vee$:

$$
\begin{aligned}
R_{\wedge,\sigma}(f, V) &= \langle \forall d{:}D_\star.\, \neg f \Rightarrow X_{false}(\star), \emptyset \rangle \\
R_{\wedge,\sigma}(X(e), V) &= \langle \forall d{:}D_\star.\, true \Rightarrow X(e), \emptyset \rangle \\
R_{\wedge,\sigma}(\varphi \wedge \psi, V) &= \langle \textstyle\bigwedge_{i \in I^\varphi \cup I^\psi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \mathcal{E}^\psi \rangle \\
R_{\wedge,\sigma}(\varphi \vee f, V) &= \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, (\neg f \wedge f_i) \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\wedge,\sigma}(\varphi, V) \\
R_{\wedge,\sigma}(f \vee \varphi, V) &= \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, (\neg f \wedge f_i) \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\wedge,\sigma}(\varphi, V) \\
R_{\wedge,\sigma}(\varphi \vee \psi, V) &= \langle \forall d{:}D_\star.\, true \Rightarrow \tilde{X}(V), (\sigma \tilde{X}(V) = \varphi \vee \psi) \rangle \\
&\qquad \text{where } \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\wedge,\sigma}(\varphi, V) \\
&\qquad\quad \langle \textstyle\bigwedge_{i \in I^\psi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\psi \rangle = R_{\wedge,\sigma}(\psi, V) \\
R_{\wedge,\sigma}(\forall W.\, \varphi, V) &= \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W \cup W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle \\
&\qquad \text{where } \langle \textstyle\bigwedge_{i \in I^\varphi} \forall W_i.\, f_i \Rightarrow X_i(g_i), \mathcal{E}^\varphi \rangle = R_{\wedge,\sigma}(\varphi, V \cup W) \\
R_{\wedge,\sigma}(\exists W.\, f, V) &= \langle \forall d{:}D_\star.\, (\exists W.\, f) \Rightarrow X_{false}(\star), \emptyset \rangle \\
R_{\wedge,\sigma}(\exists W.\, \varphi, V) &= \langle \forall d{:}D_\star.\, true \Rightarrow \tilde{X}(V), (\sigma \tilde{X}(V) = \exists W.\, \varphi) \rangle
\end{aligned}
$$

Furthermore, we define an auxiliary function $R_?$ that transforms a formula φ with

one of the rewriters $R_\vee$ or $R_\wedge$, depending on the structure of φ. This function also adds a clause with a condition that always evaluates to *true*.

$$R_{?,\sigma}(\varphi, V) = \begin{cases} \langle \varphi \vee (\exists d{:}D_\star.\ \mathit{true} \wedge X_{\mathit{false}}(\star)), \mathcal{E} \rangle \text{ where } \langle \varphi, \mathcal{E} \rangle = R_{\vee,\sigma}(\varphi, V) \\ \qquad \text{if } \varphi \text{ has the shape } \varphi_1 \vee \varphi_2, X(e) \wedge f, f \wedge X(e) \text{ or } \exists e{:}E.\ \varphi' \\ \langle \varphi \wedge (\forall d{:}D_\star.\ \mathit{true} \Rightarrow X_{\mathit{true}}(\star)), \mathcal{E} \rangle \text{ where } \langle \varphi, \mathcal{E} \rangle = R_{\wedge,\sigma}(\varphi, V) \\ \qquad \text{otherwise} \end{cases}$$

Now we can define the function R to transform a PBES:

$$\begin{aligned} R(\emptyset) &= (\mu X_{\mathit{false}}(d{:}D_\star) = X_{\mathit{false}}(\star))(\nu X_{\mathit{true}}(d{:}D_\star) = X_{\mathit{true}}(\star)) \\ R((\sigma X(V) = \varphi)\mathcal{E}) &= (\sigma X(V) = \varphi_r) R(\mathcal{E}^\varphi \mathcal{E}) \\ &\qquad \text{where } \langle \varphi_r, \mathcal{E}^\varphi \rangle = R_{?,\sigma}(\varphi, V) \end{aligned}$$

The next proposition states that SRF is a proper normal form, *i.e.*, every PBES can be transformed into SRF with the rewriter R, while preserving the solution of bound variables.

Proposition 3.3. *For every PBES $\mathcal{E}$, there is an $\mathcal{E}'$ in SRF such that $[\![\mathcal{E}]\!](X) = [\![\mathcal{E}']\!](X)$ for every $X \in \mathsf{bnd}(\mathcal{E})$.*

Proof. We sketch an inductive proof, which shows that logical equivalence of a predicate formula φ is preserved when applying one of the rewriters $R_\vee$ or $R_\wedge$. A case distinction is made for every case defined by $R_\vee$ and $R_\wedge$. For the cases that perform a recursive call to $R_\vee$ or $R_\wedge$, we assume as induction hypothesis that the resulting formula is logically equivalent to the original formula. For those cases where a new variable $\tilde{X}$ and corresponding equation are introduced, we can apply the substitution rule for PBESs [59, Lemma 18]. Termination of R can be proved by showing that every rewrite step decreases the number of disjunctive/conjunctive alternations. □

3.2 Dependency Space

The structure offered by SRF enables us to reason about the edges that must exist in proof graphs. Intuitively, a non-redundant outgoing edge from a node (X_i, v) is based on some clause $j \in J_i$ whose guard $f_j(v, e_j)$ evaluates to *true* for some value of e_j. The target node of that edge is associated to predicate variable instance $X_j(g_j(v, e_j))$. The following definition formalises this.

Definition 3.4. Let $\mathcal{E}$ be a PBES in SRF, where each equation has the same structure as in Definition 3.1. Then, the *dependency space* of $\mathcal{E}$ is a graph $G = (\mathsf{sig}(\mathcal{E}), E)$, where E is the set satisfying $(X_i, v)E(Y, w)$ for given X_i, Y, v and w iff for some $j \in J_i$ and $v_j \in \mathbb{E}_j$, we have $Y = X_j$ and both $[\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d]$ and $w = [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d]$ hold.

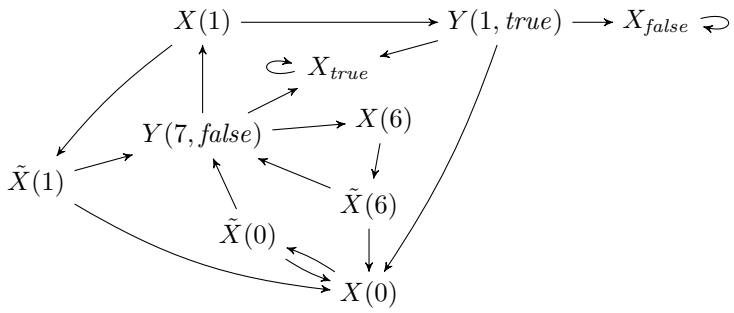


Figure 3.1: Fragment of the dependency space of the SRF-PBES from Example 3.2.

Definition 3.4 generalises the definition of a dependency space from [81], since it is applicable to all PBESs (after translation to SRF), not only to disjunctive or conjunctive PBESs.

Example 3.5. Recall again the SRF-PBES from Example 3.2. Figure 3.1 depicts the part of the dependency space that is reachable from the node $(X, 1)$. For readability, we write the node labels as $X(v)$ instead of (X, v). □

Note that every node in a dependency space has an outgoing edge; this follows from the assumption that SRF-PBESs are total. This is necessary for the validity of the next lemma.

Lemma 3.6. *The dependency space $G = (\mathsf{sig}(\mathcal{E}), E)$ of an SRF-PBES $\mathcal{E}$ is both a positive and a negative dependency graph.*

Proof. Let $G = (\mathsf{sig}(\mathcal{E}), E)$ be the dependency space of some SRF-PBES $\mathcal{E}$. Here, we prove that G is a positive dependency graph; the proof that G is a negative dependency graph is dual. For G to be a positive proof graph, we need to show for every node (X, v) that $[\![\varphi_X]\!]\eta\delta[v/d] = \mathit{true}$, with $\eta = \mathsf{env}((X, v)E, \mathit{true})$. Remark that d is the only free variable in φ_X (since $\mathcal{E}$ is closed), so we use our fixed data environment δ_0 and prove that $[\![\varphi_X]\!]\eta\delta_0[v/d] = \mathit{true}$.

Let (X_i, v) be some node of G. From the definition of $\mathsf{env}(S, \mathit{true})$ (see Definition 2.21) and the definition of a dependency space we know the following:

$$\mathsf{env}((X_i, v)E, \mathit{true})(X_j) \qquad (\dagger)$$
$$= \bigcup_{k \in J_i, v_k \in \mathbb{E}_k} \{[\![g_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d] \mid X_j = X_k \wedge [\![f_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d]\}$$

We distinguish two cases based on the right-hand side of X_i.

Case 1: the equation for X_i is conjunctive. Using the semantics of predicate formulae and (†), we can follow the reasoning below to deduce that $[\![\varphi_{X_i}]\!]\eta\delta_0[v/d] =$

true, where $\eta = \mathsf{env}((X_i, v)E, \mathit{true})$.

$$
\begin{aligned}
[\![\varphi_{X_i}]\!]\eta\delta_0[v/d] &= [\![\bigwedge_{j \in J_i} \forall e_j{:}E_j.\, f_j(d, e_j) \Rightarrow X_j(g_j(d, e_j))]\!]\eta\delta_0[v/d] \\
&= \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\eta\delta_0[v_j/e_j, v/d] \Rightarrow [\![X_j(g_j(d, e_j))]\!]\eta\delta_0[v_j/e_j, v/d] \\
&= \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \Rightarrow [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \in \eta(X_j) \\
&\overset{(\dagger)}{=} \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \Rightarrow [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \in \\
&\qquad \bigcup_{k \in J_i, v_k \in \mathbb{E}_k} \{[\![g_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d] \mid X_j = X_k \wedge [\![f_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d]\} \\
&\Leftarrow \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \Rightarrow [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \in \\
&\qquad \bigcup_{v'_j \in \mathbb{E}_j} \{[\![g_j(d, e_j)]\!]\delta_0[v'_j/e_j, v/d] \mid [\![f_j(d, e_j)]\!]\delta_0[v'_j/e_j, v/d]\} \\
&= \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \Rightarrow \\
&\qquad \exists v'_j \in \mathbb{E}_j.\, v'_j = v_j \wedge [\![f_j(d, e_j)]\!]\delta_0[v'_j/e_j, v/d] \\
&= \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \Rightarrow [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \\
&= \bigwedge_{j \in J_i} \forall v_j \in \mathbb{E}_j.\, \mathit{true} \\
&= \mathit{true}
\end{aligned}
$$

Case 2: the equation for X_i is disjunctive. We again use the semantics of predicate formulae and (†), and follow a similar reasoning as before.

$$
\begin{aligned}
[\![\varphi_{X_i}]\!]\eta\delta_0[v/d] &= [\![\bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge X_j(g_j(d, e_j))]\!]\eta\delta_0[v/d] \\
&= \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\eta\delta_0[v_j/e_j, v/d] \wedge [\![X_j(g_j(d, e_j))]\!]\eta\delta_0[v_j/e_j, v/d] \\
&= \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \wedge [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \in \eta(X_j) \\
&\overset{(\dagger)}{=} \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \wedge [\![g_j(d, e_j)]\!]\delta_0[v_j/e_j, v/d] \in \\
&\qquad \bigcup_{k \in J_i, v_k \in \mathbb{E}_k} \{[\![g_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d] \mid X_j = X_k \wedge [\![f_k(d, e_k)]\!]\delta_0[v_k/e_k, v/d]\}
\end{aligned}
$$

$$
\begin{aligned}
&\Leftarrow \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d] \wedge [\![g_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d] \in \\
&\qquad \bigcup_{v'_j \in \mathbb{E}_j} \{[\![g_j(d,e_j)]\!]\delta_0[v'_j/e_j, v/d] \mid [\![f_j(d,e_j)]\!]\delta_0[v'_j/e_j, v/d]\} \\
&= \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d] \wedge \\
&\qquad \exists v'_j \in \mathbb{E}_j.\, v'_j = v_j \wedge [\![f_j(d,e_j)]\!]\delta_0[v'_j/e_j, v/d] \\
&= \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d] \wedge [\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d] \\
&= \bigvee_{j \in J_i} \exists v_j \in \mathbb{E}_j.\, [\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d]
\end{aligned}
$$

From assumption that $\mathcal{E}$ is total, we have that for every node (X, v), at least one condition f_j evaluates to *true* (cf. Definition 3.1), we conclude that $[\![\varphi_{X_i}]\!]\eta\delta_0[v/d] = \mathit{true}$. With this, the condition on transitions in a positive dependency graph is satisfied. □

Theorem 3.7. *The dependency space of an SRF-PBES $\mathcal{E}$ is the unique smallest dependency graph with $V = \mathsf{sig}(\mathcal{E})$ that is both positive and negative.*

Proof. By contradiction. Let $G = (\mathsf{sig}(\mathcal{E}), E)$ be the dependency space for some SRF-PBES $\mathcal{E}$ and let $G' = (\mathsf{sig}(\mathcal{E}), E')$ be a dependency graph that is both positive and negative such that $E \not\subseteq E'$, *i.e.*, G is not a subgraph of G'. That means that there is at least one edge in E that is missing from E'. Let $(X, v)E(Y, w)$ be such an edge. From the definition of a dependency space, we can deduce that $Y = X_j$ for some $j \in J_i$. Furthermore, for some $v_j \in \mathbb{E}_j$, the condition $[\![f_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d]$ holds and $[\![g_j(d,e_j)]\!]\delta_0[v_j/e_j, v/d]$ has value w. Therefore, (X, v) depends on (Y, w) in one of two ways:

- In case the equation for X is conjunctive, (Y, w) necessarily has to hold in order for (X, v) to hold. This is not reflected by G', which is therefore not a positive dependency graph.
- In case the equation for X is disjunctive, (Y, w) necessarily has to be false in order for (X, v) to be false. This is not reflected by G', which is consequently not a negative dependency graph.

We conclude that G' is either not a positive or not a negative dependency graph, which contradicts our initial assumption. □

We conclude from Lemma 3.6 and Theorem 3.7 that a dependency space exactly captures all positive and negative dependencies.

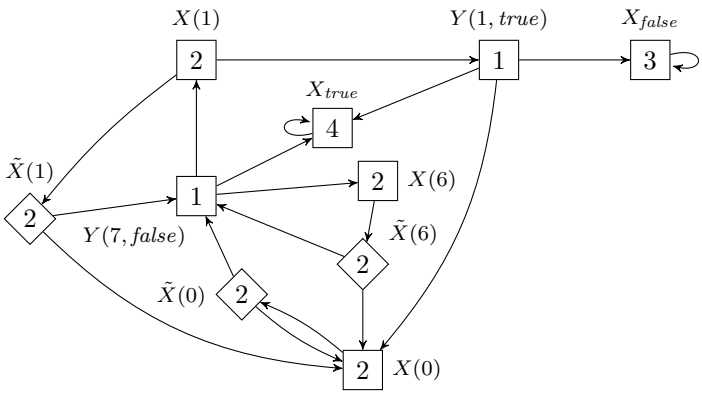


Figure 3.2: Parity game corresponding to the SRF-PBES of Example 3.2.

3.3 Relation to Parity Games

Similar to parity games, every node in a dependency space has an associated operator (player) and rank (priority). Furthermore, the infinite path condition of a proof graph corresponds to the winning condition of infinite plays in a parity game. This indicates that there is a close relation between dependency spaces and parity games. We investigate this relationship in this section.

Definition 3.8. Let $G = (V, E)$ be the dependency space of an SRF-PBES $\mathcal{E}$. The parity game corresponding to $\mathcal{E}$ is $(V, E, \Omega, \mathcal{P})$, where:

- $\Omega((X, v)) = \mathsf{rank}_{\mathcal{E}}(X)$; and
- $\mathcal{P}((X, v)) = \Diamond$ if and only if $\mathsf{op}_{\mathcal{E}}(X) = \vee$.

To demonstrate the construction, we revisit our running example.

Example 3.9. We take the dependency space from Example 3.5 and construct the parity game that corresponds to the SRF-PBES of Example 3.2. Again, we only consider the nodes that are reachable from $(X, 1)$; this subgame is depicted in Figure 3.2. Player $\Box$ wins nodes X_{false}, $Y(1, true)$, $X(1)$ and $Y(7, false)$; player $\Diamond$ wins the other nodes. $\Box$

It turns out that, given a PBES $\mathcal{E}$, the existence of a positive (resp. negative) proof graph within the dependency space of $\mathcal{E}$ coincides with the existence of a winning strategy for $\Diamond$ (resp. $\Box$) in the parity game that corresponds to $\mathcal{E}$. This is formalised in the next theorem.

Theorem 3.10. *Let $\mathcal{E}$ be a PBES in SRF, $G = (V, E)$ its dependency space and $G' = (V, E, \Omega, \mathcal{P})$ the corresponding parity game. Then, $v \in [\![\mathcal{E}]\!](X)$ iff (X, v) is won by player $\Diamond$ in G'.*

Proof. Let $\mathcal{E}$, G and G' be as above and let (X, v) be an arbitrary node in G'. Here, we only consider the case where (X, v) is won by player $\diamond$, the other case is analogous. Let σ be a winning strategy for $\diamond$ in (X, v) and let U be some σ-dominion that contains (X, v). Consider the subgame $G' \cap U \cap \sigma = (U, E_\sigma \cap (U \times U), \Omega|_U, \mathcal{P}|_U)$. The corresponding proof graph is $G_\sigma = (U, E_\sigma \cap (U \times U))$, which is a subgraph of G. We show that G_σ is indeed a positive proof graph. Since conjunctive nodes have the same successors in G and G_σ and disjunctive nodes have at least a single successor of which the condition evaluates to *true* (by Definition 3.4), we can apply the same reasoning as in the proof of Lemma 3.6 to conclude that the dependency graph condition (see Definition 2.21) is satisfied for G_σ. All paths in G_σ are by definition consistent with σ. From the fact that σ is a winning strategy for $\diamond$, it follows that for all paths in G_σ, the minimal priority that occurs infinitely often is even. Consequently, the proof graph condition (Definition 2.23) is also satisfied for G_σ.

We conclude that G_σ is a positive proof graph that contains (X, v), hence we also have $v \in [\![\mathcal{E}]\!](X)$ by Theorem 2.25. □

Now that we have defined the relation between PBESs and parity games, we review the two model checking work flows LPS-LTS-PG and LPS-PBES-PG (see also Figure 2.6). Although the two resulting parity games have the same winner (Theorems 2.15, 2.25, 2.27 and 3.10), the games are not always isomorph. This is due to the fact that the construction of a parity game from an LTS and a μ-calculus formula completely unfolds the formula into new parity game nodes, regardless of whether an alternation between a conjunctive and disjunctive subformula occurs. On the other hand, in an SRF-PBES, a conjunctive right-hand side may encode multiple conjunctive subformulae. Other subtle differences are that (i) the games following from the PBES route are always total, while those constructed through an LTS are not; and (ii) both games can have slightly different priorities (but these are never relevant for the solution).

3.4 Clustered Recursive Form

Standard recursive form offers sufficient structure to construct a dependency space. However, the fact that predicate variables may occur multiple times in a given right-hand side requires us to reason about the equality $Y = X_j$ and makes the notation as well as some of the proofs of theorems in this thesis slightly more complex. To further simplify the notation and the reasoning about dependencies, we introduce *clustered recursive form* for PBESs.

Definition 3.11. A predicate formula is in *clustered recursive form* (CRF) iff it is disjunctive or conjunctive and the predicate variable in each of the clauses is unique, *i.e.*, $X_j \neq X_k$ for all distinct $j, k \in J$. A PBES is in CRF iff all its right-hand sides are CRF formulae.

Similar to SRF-PBESs, we henceforward assume that every CRF-PBES is total. For CRF-PBESs, the question whether (X_i, v) depends on (X_j, w) only requires

deciding

$$\exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j] \wedge w = [\![g_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j]$$

We observe that every PBES $\mathcal{E}$ can be transformed to CRF by first rewriting $\mathcal{E}$ to SRF and subsequently combining clauses that have the same predicate variable, relying on suitable projection operators for the data arguments.

Example 3.12. Consider again the following equation from the SRF-PBES of Example 3.2.

$$\begin{aligned}\mu Y(n{:}N, b{:}B) = {} & (b \Rightarrow X_{false}) \\ & \wedge (n > 0 \Rightarrow X(n-1)) \\ & \wedge (n > 5 \Rightarrow X(1)) \\ & \wedge (\mathit{true} \Rightarrow X_{true})\end{aligned}$$

The second and third clause can be combined by encoding the choice between the two in a universally quantified Boolean variable e:

$$\begin{aligned}\mu Y(n{:}N, b{:}B) = {} & (b \Rightarrow X_{false}) \\ & \wedge \forall e{:}B.\, (\mathit{if}(e, n > 0, n > 5) \Rightarrow X(\mathit{if}(e, n-1, 1))) \\ & \wedge (\mathit{true} \Rightarrow X_{true})\end{aligned}$$

□

This technique is formalised in the next proposition and corollary.

Proposition 3.13. *For every PBES $\mathcal{E}$ in SRF, there is a PBES $\mathcal{E}'$ in CRF such that $[\![\mathcal{E}]\!](X) = [\![\mathcal{E}']\!](X)$ for every $X \in \mathsf{bnd}(\mathcal{E})$.*

Proof. Let $\mathcal{E}$ be a PBES in SRF. Furthermore, let $(\sigma_i X_i(d{:}D) = \varphi_{X_i}) \in \mathcal{E}$ be some equation and $Y \in \mathsf{bnd}(\mathcal{E})$ a predicate variable that occurs multiple times in φ_{X_i}. We consider the case that φ_{X_i} is disjunctive, *i.e.*, it is of the shape $\bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge X_j(g_j(d, e_j))$. The proof for the conjunctive case is analogous. We consider the set of clauses that contain Y; their indices are $\{j_1, \ldots, j_n\} \subseteq J_i$. More formally, $Y = X_{j_k}$ for all $1 \leq k \leq n$ and $Y \neq X_j$ for all $j \in J_i \setminus \{j_1, \ldots, j_n\}$. Let $D_n = \{1, \ldots, n\}$ be a sort with n elements and pr_n a polymorphic projection function that has signature $D_n \times T^n \to T$ for any type T; it is defined as $pr_n(k, t_1, \ldots, t_k, \ldots, t_n) = t_k$ for all $k{:}D_n$. Then, the n clauses for Y can be grouped in one clause as follows:

$$\begin{aligned}\exists k{:}D_n, e_{j_1}{:}E_{j_1}, \ldots, e_{j_n}{:}E_{j_n}.\, & pr_n(k, f_{j_1}(d, e_{j_1}), \ldots, f_{j_n}(d, e_{j_n})) \\ & \wedge Y(pr_n(e_n, g_{j_1}(d, e_{j_1}), \ldots, g_{j_n}(d, e_{j_n})))\end{aligned}$$

Unfolding the existential quantifier over D_n, applying the definition of pr_n and eliminating unused quantified variables, results in exactly the original set of clauses, so this transformation preserves the solution of $\mathcal{E}$. By applying the same construction to all predicate variables that occur in multiple clauses of the same equation, $\mathcal{E}$ can be rewritten to CRF. □

Corollary 3.14. *For every PBES $\mathcal{E}$, there is an $\mathcal{E}'$ in CRF such that $[\![\mathcal{E}]\!](X) = [\![\mathcal{E}']\!](X)$ for every $X \in \mathsf{bnd}(\mathcal{E})$.*

Proof. Follows from Propositions 3.3 and 3.13. □

3.5 Related Work

In the literature, several other normal forms for BESs or PBESs have been proposed. Firstly, our SRF for PBESs generalises SRF for BESs, which was proposed in [75]; a BES is in SRF if and only if every right-hand side is of the form

$$\bigvee_{i \in I} X_i(\star) \qquad \text{or} \qquad \bigwedge_{i \in I} X_i(\star)$$

If the right-hand sides are disjunctive or conjunctive (as above), but the constants *true* and *false* occur, then the BES is in *simple form* [93]. All BESs can be transformed to simple form or SRF in polynomial time, while introducing a number of new equations that is no greater than the sum of the sizes of all right-hand sides.

A PBES is in *predicate formula normal form* (PFNF) [106] if and only if every right-hand side has the shape

$$\mathsf{Q}_1 d_1{:}D_1 \ldots \mathsf{Q}_n d_n{:}D_n.\, h \wedge \bigwedge_{i \in I} \Big(g_i \Rightarrow \bigvee_{j \in J_i} X_j(e_j) \Big)$$

where every $\mathsf{Q}_i \in \{\exists, \forall\}$, I is a, possibly empty, finite index set, each J_i is a non-empty index set and h and g_i are Boolean terms. Every predicate formula can be translated to an equivalent PFNF formula, although this can cause a quadratic blow-up. The blow-up can be avoided by introducing new PBES equations.

Koolen *et al.* [81] use *disjunctive normal form* (DNF) as a standard representation for disjunctive predicate formulae, in which predicate variables can only occur in the context of a conjunction if the other operand is a simple formula. Finally, the *parameterised parity game* (PPG) form of [72] is similar to our standard recursive form, but does not guarantee that all dependency graphs have a total transition relation. Totality of the transition relation is a necessary condition for the existence of a dependency space. The translation from arbitrary PBESs to PPG relies on an intermediate format called *bounded quantifier normal form* (BQNF), which is related to PFNF, but no translation from PBES to BQNF is given in [72].

3.6 Conclusion

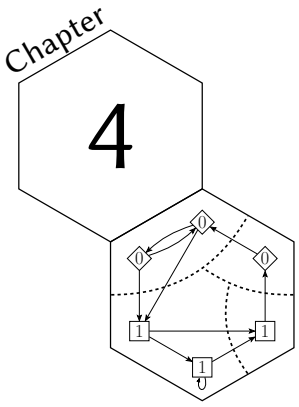
We have introduced the standard and clustered recursive forms for arbitrary PBESs. Unlike many existing normal forms, SRF and CRF eliminate alternations between disjunctive and conjunctive operators within every right-hand side. This can require introducing new equations, but the total size of the PBES does not grow significantly.

In some cases the SRF transformation may increase the size of the underlying dependency graphs. For example, the PBES $(\nu X = \forall m{:}N.\, \exists n{:}N.\, (n = m \wedge Y))(\mu Y = Y)$ has only finite dependency graphs, while the corresponding SRF-PBES $(\nu X = \forall m{:}N.\, \tilde{X}(m))(\nu \tilde{X}(m{:}N) = \exists n{:}N.\, n = m \wedge Y)(\mu Y = Y)$ also has infinite dependency graphs. In practice, most PBESs that encode a model checking problem are already very close to being in SRF, since a major part of the right-hand sides stems from μ-calculus subformulae of the shape $[\alpha]X$ or $\langle\alpha\rangle X$. The corresponding predicate formula in the PBES is, by Definition 2.26, conjunctive or disjunctive. As a result, this issue does not occur in the encoding of most reasonable μ-calculus formulae.

From a PBES in SRF, one can construct a dependency space, which captures both positive and negative dependencies. A dependency space can be interpreted as a parity game; the existence of a proof graph in the dependency space coincides with the existence of a winning strategy in the corresponding parity game. The structure offered by SRF and CRF plays a fundamental role in the approaches of Chapters 4 and 6.

Chapter 4

Symbolic Bisimulation for PBESs



Although finding the solution of a PBES is undecidable in general, in practice several efficient approaches to solve PBESs exist. Most notably, some PBESs can be solved efficiently by first simplifying them—if needed—using static analysis techniques [106, 74], instantiating them to finite *Boolean equation systems* (BESs) and subsequently solving these BESs [114]. However, for many types of problems, the corresponding PBES contains data taken from domains that are infinite. For example, a PBES encoding the mutual exclusion property for Lamport's bakery protocol requires data variables ranging over natural numbers. Similarly, PBESs encoding model checking problems for timed or hybrid systems, typically modelled by timed automata or hybrid automata, contain data variables that range over real numbers.

Several symbolic techniques have been proposed to deal with PBESs over infinite data domains [100, 81, 47], but their application is unfortunately limited to specific subclasses of PBESs. Typically, these fragments exclude PBESs in which both types of logical quantifier occur; *i.e.*, PBESs may only contain universal quantification or only existential quantification. Such constraints effectively limit the class of properties that can be encoded, excluding, *e.g.*, most behavioural equivalence decision problems, but also many CTL* properties.

This chapter explores an approach, called *PBES quotienting*, that is more general than the symbolic techniques discussed above: PBES quotienting is applicable to the

full class of PBESs. Our approach is based on *minimal model generation* (MMG) [23], a similar procedure that operates on behavioural models, such as extended finite state machines (which are very similar to LPSs) or timed automata. PBES quotienting relies on the CRF normal form and the parity game that can be derived through it; both these concepts were introduced in Chapter 3. A procedure based on quotienting can be used to compute a minimal reduced dependency graph from a symbolic representation of the dependency space. As we will see, PBES quotienting can be further improved by extracting finite partial solutions from PBESs that have an infinite minimal reduced dependency space.

To validate the above, we perform a number of experiments with an implementation of our procedures and compare these to the solver of [81]. Furthermore, we experimentally compare our PBES techniques with an implementation of MMG. The results of this evaluation show that our technique is indeed capable of solving decision problems that existing approaches fail to solve so far. In particular, the experiments show that PBES quotienting is a promising generic approach for model checking of (timed) modal μ-calculus properties on systems with infinite data domains and also equivalence checking of systems with infinite data domains.

The rest of the chapter is structured as follows: Section 4.1 introduces the ideas of minimal model generation. Section 4.2 contains an example that shows how minimal model generation can be applied and what its shortcomings are. Then, Section 4.3 shows how the minimal model generation procedure can be adapted to the setting of PBESs. Two improvements to the procedure are presented in Sections 4.4 and 4.5 respectively. In Section 4.6, we perform experiments to compare minimal model generation with the new PBES procedure, and its optimisations. Finally, Section 4.7 gives an overview of related work and Section 4.8 presents a conclusion and suggestions for future work.

4.1 Minimal Model Generation

The most common and straightforward way of analysing the behaviour specified by an LPS is to construct the corresponding transition system by means of *state space exploration*. Starting from the initial state, the exploration procedure computes outgoing transitions and stores new states it encounters. However, this technique is not complete: for processes with an infinite reachable state space, the procedure does not terminate. This is demonstrated in the following example.

Example 4.1. We consider a model of a primitive coffee machine that accepts coins of 5 cents and 10 cents and never gives change. After ordering a coffee and inputting at least 15 cents, the machine can give coffee. A possible representation of this machine

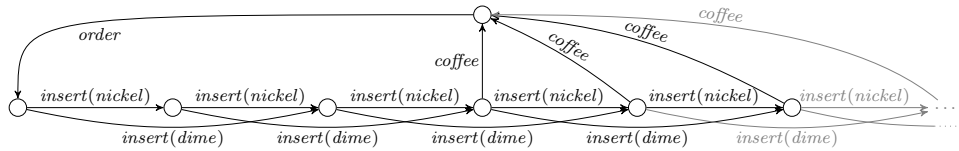


Figure 4.1: Infinite LTS of the coffee machine of Example 4.1.

is the LPS $(\mathit{Machine}, (\mathit{false}, 0))$, where $\mathit{Machine}$ is the process defined as follows.

$$\begin{aligned}\mathit{Machine}(&\mathit{idle}{:}B, \mathit{balance}{:}N) = \\ &\mathit{idle} \to \mathit{order} \cdot \mathit{Machine}(\mathit{false}, \mathit{balance}) \\ &+ \sum_{c:\mathit{Coin}} \neg\mathit{idle} \to \mathit{insert}(c) \cdot \mathit{Machine}(\mathit{false}, \mathit{balance} + \mathit{value}(c)) \\ &+ (\neg\mathit{idle} \wedge \mathit{balance} \geq 15) \to \mathit{coffee} \cdot \mathit{Machine}(\mathit{true}, 0)\end{aligned}$$

Here, the parameter *idle* expresses whether the machine is idle or a transaction is in process and *balance* stores the amount of money inserted during the current transaction. Furthermore, *Coin* is a data sort containing the values *nickel* and *dime* and $\mathit{value}\colon \mathit{Coin} \to N$ computes the corresponding value of a coin. Part of the reachable state space of the associated LTS is depicted in Figure 4.1. Since there is no upper bound on the amount of money that can be inserted, the state space is infinite. □

To deal with instances like the coffee machine above, many symbolic approaches have been developed, one of which is *minimal model generation* (MMG) [23, 46], which we discuss below. Before we can introduce the MMG approach, we first introduce two more concepts that make up the underlying theory. First, we introduce the concept of a reduced LTS, which allows us to reason about certain infinite LTSs using a finite representation. Reduced LTSs rely on the notion of a *partition*: a set $A \subseteq 2^B$ is a partition of a set B if and only if $\bigcup A = B$ and for distinct $a, a' \in A$, it holds that $a \cap a' = \emptyset$.

Definition 4.2. Let $TS = (S, \to, \hat{s})$ be an LTS. Then, $TS_r = (S_r, \to_r, b)$ is a *reduced LTS* iff:

- $S_r \subseteq 2^S$ is a partition of S;
- $\to_r = \{(b, a, b') \mid \exists s \in b, t \in b' .\, s \xrightarrow{a} t\}$.

We say TS is the *base LTS* of TS_r.

We commonly refer to an element $b \in S_r$ as a *block*. Remark that the set of transitions between two blocks is an over-approximation of the transitions of the constituting states. Hence, not all reduced LTSs are a meaningful representation of

their base LTS. After all, any LTS can be represented by a reduced LTS with only one block that contains all states. To preserve interesting properties, every block should only contain states that are related by a certain equivalence relation. Here, we use *bisimulation* [109], which preserves most logic properties, such as those formulated in the modal μ-calculus.

Definition 4.3. Given an LTS $TS = (S, \rightarrow, \hat{s})$, a relation $\mathcal{R} \subseteq S \times S$ on states is a *bisimulation relation* iff for all s, t such that $s\mathcal{R}t$, it holds that:

- For all transitions $s \xrightarrow{a} s'$, there is a transition $t \xrightarrow{a} t'$ such that $s'\mathcal{R}t'$.
- For all transitions $t \xrightarrow{a} t'$, there is a transition $s \xrightarrow{a} s'$ such that $s'\mathcal{R}t'$.

We say two states s and t are *bisimilar*, notation $s \leftrightarroweq t$, iff they are related by some bisimulation relation. Two LTSs are bisimilar iff their initial states are bisimilar.

We call the reduced LTS that is minimal under bisimulation the *bisimulation quotient*, which we denote with ${}^{TS}/_{\leftrightarroweq}$. The state space of ${}^{TS}/_{\leftrightarroweq}$, denoted ${}^{S}/_{\leftrightarroweq}$, consists of the equivalence classes induced by bisimulation, *i.e.*, states s and t are in the same block if and only if they are bisimilar. Observe that the bisimulation quotient is well-defined.

Partition refinement To compute the bisimulation quotient, we rely on *partition refinement*. In this procedure, a partition of the state space is iteratively refined until it becomes *stable* (a formal definition follows). We say a partition π is *finer* than a partition π' iff all blocks of π are contained in some block of π'. The coarsest stable partition coincides with the equivalence classes under bisimulation.

Procedure 1: Minimal model generation for LPSs

Input: LPS $L = (P, \hat{d})$, **initial partition** π_0

1 $n := 0$;
2 **while** π_n *is not stable* **do**
3 $\quad n := n + 1$;
4 $\quad \pi_n := (\pi_{n-1} \setminus \{b\}) \cup \{\mathit{split}(b, b', a), \mathit{co\text{-}split}(b, b', a)\}$
$\qquad$ **for some** $b, b' \in \pi_{n-1}$, $a \in \mathit{Act}$
$\qquad$ **such that** $\mathit{split}(b, b', a)$ and $\mathit{co\text{-}split}(b, b', a)$ are non-empty;
5 $\quad \rightarrow_n := \{(b, a, b') \mid \exists s \in b, t \in b'.\, s \xrightarrow{a} t\}$;
6 $\quad \pi_n := \{b \in \pi_n \mid \hat{b} \rightarrow_n^* b\}$ **where** $[\![\hat{d}]\!] \in \hat{b}$;
7 **return** $(\pi_n, \rightarrow_n, \hat{b})$ **where** $[\![\hat{d}]\!] \in \hat{b}$;

Procedure 1 shows how to perform partition refinement on an LTS $TS = (\mathbb{D}, \rightarrow, \hat{v})$ that underlies the LPS L. The initial partition has one block containing all states, *i.e.*, $\pi_0 = \{\mathbb{D}\}$. In every iteration, we find two blocks $b, b' \in \pi_n$ and an action a and

split b with respect to b' in the following way:

$$\begin{aligned} \mathit{split}(b, b', a) &= \{s \in b \mid \exists t \in b'.\, s \xrightarrow{a} t\} \\ \mathit{co\text{-}split}(b, b', a) &= b \setminus \mathit{split}(b, b', a) \end{aligned}$$

Then we update the partition and transition relation to reflect this split (lines 4 and 5). Next, we use a simple forward exploration on blocks to compute which blocks are reachable from the initial block, and discard blocks that are not reachable (line 6). Here, $\rightarrow_n^*$ is the reflexive transitive closure of the transition relation. Strictly speaking, after discarding one or more blocks, π_n is no longer a partition of the complete state space, but only of some over-approximation of the reachable state space. Note that each partition π_{n+1} is finer than partition π_n.

If a block b cannot be split with respect to a block b' for all actions a, we say b is *stable* (under bisimulation) with respect to b'. Block b is stable with respect to a set of blocks K iff it is stable with respect to all the blocks in K. A partition π is stable (with respect to itself) iff all of the blocks in π are stable with respect to π. The partition refinement procedure terminates when π is stable (line 2). The reachable part of the bisimulation quotient can be constructed from the stable partition using Definition 4.2. Remark that termination is not guaranteed as not every infinite LTS has a finite bisimulation quotient. Consequently, Procedure 1, and also the other procedures we present below, is a semi-decision procedure.

Since our goal is to enable reasoning about LTSs with an infinite state space, we cannot store blocks by storing each of their constituent states explicitly. Instead, we represent each block with a *characteristic function*.

Definition 4.4. Let L be an LPS and b be a set of states in the associated LTS. The corresponding *characteristic function* $\mathbb{K}_b : \mathbb{D} \to \mathbb{B}$ is defined as:

$$\mathbb{K}_b(v) = \begin{cases} \mathit{true} & \text{if } v \in b \\ \mathit{false} & \text{otherwise} \end{cases}$$

Henceforth, we represent the semantic function $\mathbb{K}_b$ for block b with a syntactic Boolean expression k_b. With this representation, we can implement Procedure 1 symbolically, see Procedure 2.

Firstly, the initial partition π_0 is represented by $\{\mathit{true}\}$. Secondly, $\mathit{split}(k, k', a)$ and $\mathit{co\text{-}split}(k, k', a)$ are the respective symbolic implementations of $\mathit{split}(b, b', a)$ and $\mathit{co\text{-}split}(b, b', a)$. Recall that a linear process has the shape $P(d{:}D) = \sum_{i \in I} \sum_{e_i : E_i} c_i(d, e_i) \rightarrow a_i(f_i(d, e_i)) \cdot P(g_i(d, e_i))$. In the following definitions, k and k' are the characteristic functions corresponding to blocks b and b', respectively. Furthermore, we assume every action has the shape $a(e)$, where a is an action label, and e is the

Procedure 2: Symbolic minimal model generation for LPSs

Input: LPS $L = (P, \hat{d})$, **initial partition** π_0

1 $n := 0$;
2 **while** π_n *is not stable* **do**
3 $\quad n := n + 1$;
4 $\quad \pi_n := (\pi_{n-1} \setminus \{k\}) \cup \{split(k, k', a), co\text{-}split(k, k', a)\}$
$\qquad$ **for some** $k, k' \in \pi_{n-1}$, $a \in Act$
$\qquad$ **such that** $split(k, k', a)$ and $co\text{-}split(k, k', a)$ are not equivalent to *false*;
5 $\quad \rightarrow_n := \{(k, a, k') \mid \exists v \in \mathbb{D}.\, [\![k(d)]\!]\delta_0[v/d] \wedge \bigvee_{i \in I} \exists v_i \in \mathbb{E}_i.\, [\![c_i(d, e_i)]\!]\delta_0[v/d, v_i/e_i]$
$\qquad \wedge\, a = a_i([\![f_i(d, e_i)]\!]\delta_0[v/d, v_i/e_i]) \wedge [\![k'(g_i(d, e_i))]\!]\delta_0[v/d, v_i/e_i]\}$;
6 $\quad \pi_n := \{k \in \pi_n \mid \hat{k} \rightarrow_n^* k\}$ **where** $\hat{k} \in \pi_n$ **such that** $[\![\hat{k}(\hat{d})]\!]\delta_0$ holds;
7 **return** $(\pi_n, \rightarrow_n, \hat{k})$ **where** $[\![\hat{k}(\hat{d})]\!]\delta_0$ holds;

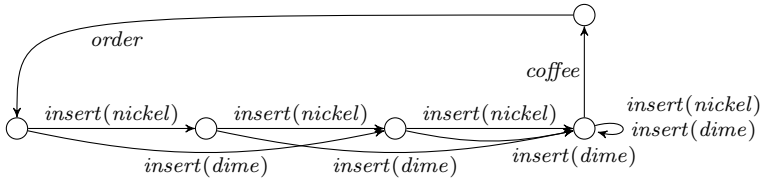


Figure 4.2: The bisimulation quotient of the coffee machine with an infinite state space (Example 4.1) as computed by Procedure 2.

action parameter, expressed as a ground term.

$$split(k, k', a(e)) = k(d) \wedge \bigvee_{i \in I} \exists e_i{:}E_i.\, \big(c_i(d, e_i) \wedge a(e) = a_i(f_i(d, e_i)) \wedge k'(g_i(d, e_i))\big)$$

$$co\text{-}split(k, k', a(e)) = k(d) \wedge \neg \bigvee_{i \in I} \exists e_i{:}E_i.\, \big(c_i(d, e_i) \wedge a(e) = a_i(f_i(d, e_i)) \wedge k'(g_i(d, e_i))\big)$$

A similar symbolic implementation is necessary to compute the transition relation at line 5 of Procedure 2, where we compute $\exists s \in b, t \in b'.\, s \xrightarrow{a} t$ symbolically.

Example 4.5. We revisit the coffee machine with an infinite state space from Example 4.1. The bisimulation quotient generated by Procedure 2 is depicted in Figure 4.2. All states reached by inserting more than 15 cents are collapsed into one (the state with the self loop), since they are all bisimilar. A characteristic function for this block can be $\neg idle \wedge balance \geq 15$. □

Although characteristic functions help to deal with an infinite state space, minimal

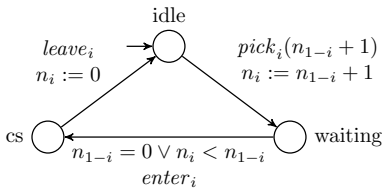


Figure 4.3: Process i from the simplified bakery protocol.

model generation still does not deal with the case where the domain of action parameters is infinite. This is due to the fact that it is still necessary to iterate over all actions on line 4 of Procedure 2. The next section presents an example that shows how these limitations impact the ability to perform model checking using minimal model generation.

4.2 Motivating Example

To show how MMG can be used for model checking and illustrate its limitations, we introduce a slightly larger example in this section. This example will also serve as a running example throughout the rest of the chapter. The model we consider is a simplified version of Lamport's bakery protocol [86]. In our setting, there are only two processes (customer 0 and customer 1) and all writes and reads are atomic. When customer i enters the bakery, he/she does not have a number ($n_i = 0$). At any point, the customer can pick a number, which is one larger than the number of the other customer. If both customers are waiting, the customer with the smallest number can enter the critical section. When leaving the critical section, the number is discarded (n_i is reset to 0). See Figure 4.3.

The LPS $L = (Bakery, (idle, 0, idle, 0))$ represents the behaviour of the two customers, where *Bakery* is the following linear process.

$$
\begin{aligned}
&Bakery(s_0{:}State, n_0{:}N, s_1{:}State, n_1{:}N) = \\
&\qquad (s_0 = idle) \to pick_0(n_1+1) \cdot Bakery(waiting, n_1+1, s_1, n_1) \\
&\quad + (s_0 = waiting \wedge (n_1 = 0 \vee n_0 < n_1)) \to enter_0 \cdot Bakery(cs, n_0, s_1, n_1) \\
&\quad + (s_0 = cs) \to leave_0 \cdot Bakery(idle, 0, s_1, n_1) \\
&\quad + (s_1 = idle) \to pick_1(n_0+1) \cdot Bakery(s_0, n_0, waiting, n_0+1) \\
&\quad + (s_1 = waiting \wedge (n_0 = 0 \vee n_1 < n_0)) \to enter_1 \cdot Bakery(s_0, n_0, cs, n_1) \\
&\quad + (s_1 = cs) \to leave_1 \cdot Bakery(s_0, n_0, idle, 0)
\end{aligned}
$$

In this encoding, s_i and n_i represent the state and number of customer i, respectively. Furthermore, the states of a single process are encoded in the sort *State*.

On this model, we would like to check the property "regardless of which number customer 0 picks, he/she can always enter the critical section in a finite time". This

can be formalised with the modal μ-calculus formula

$$\nu X.\,([\top]X \wedge \forall m{:}N.\,[pick_0(m)]\mu Y.\,([\,\overline{enter_0}\,]Y \wedge \langle\top\rangle true))$$

Here, the fixpoint variable X ranges over the whole state space and Y holds if and only if for all paths, $enter_0$ occurs within a finite number of steps. However, the state space of the LPS is infinite and also the set of actions is infinite, due to the parameter of the $pick_i$ actions, which is a natural number. Furthermore, the bisimulation quotient is also infinitely large. Therefore, neither classical state space exploration nor minimal model generation can compute an LTS on which the formula can be evaluated.

An alternative approach is to encode this model checking question in a parameterised Boolean equation system. Taking the LPS above and the formula, the following PBES can be constructed according to Definition 2.26:

$$\begin{aligned}
&\nu X(s_0{:}S, n_0{:}N, s_1{:}S, n_1{:}N) = \\
&\qquad s_0 = idle \Rightarrow Y(n_1 + 1, s_1, n_1) && (1)\\
&\quad \wedge\, s_0 = idle \Rightarrow X(waiting, n_1 + 1, s_1, n_1) && (2)\\
&\quad \wedge\, s_0 = waiting \wedge (n_1 = 0 \vee n_0 < n_1) \Rightarrow X(cs, n_0, s_1, n_1) && (3)\\
&\quad \wedge\, s_0 = cs \Rightarrow X(idle, 0, s_1, n_1) && (4)\\
&\quad \wedge\, s_1 = idle \Rightarrow X(s_0, n_0, waiting, n_0 + 1) && (5)\\
&\quad \wedge\, s_1 = waiting \wedge (n_0 = 0 \vee n_1 < n_0) \Rightarrow X(s_0, n_0, cs, n_1) && (6)\\
&\quad \wedge\, s_1 = cs \Rightarrow X(s_0, n_0, idle, 0) && (7)\\
&\mu Y(n_0{:}N, s_1{:}S, n_1{:}N) = \\
&\qquad ((n_1 = 0 \vee n_0 < n_1) \vee && (8)\\
&\qquad\quad s_1 = idle \vee (s_1 = waiting \wedge (n_0 = 0 \vee n_1 < n_0)) \vee s_1 = cs) && (9)\\
&\quad \wedge\, s_1 = idle \Rightarrow Y(n_0, waiting, n_0 + 1) && (10)\\
&\quad \wedge\, s_1 = waiting \wedge (n_0 = 0 \vee n_1 < n_0) \Rightarrow Y(n_0, cs, n_1) && (11)\\
&\quad \wedge\, s_1 = cs \Rightarrow Y(n_0, idle, 0) && (12)
\end{aligned}$$

Predicate variable X represents the fact that the property has to hold at any point in time. Therefore, it is labelled with a greatest fixpoint and it encodes the full behaviour of the system in a way very similar to the *Bakery* LPS (lines 2 to 7). When customer 0 picks a number, we check the second half of the property using Y (line 1). For predicate variable Y, we assume that customer 0 is in the state *waiting*. Then, Y is *true* if customer 0 can enter the critical section (line 8) or customer 1 does something else after which Y holds (line 9 and lines 10 to 12). However, customer 1 is only allowed to do something finitely often, so the equation for Y is labelled with a least fixpoint. The property holds, since the solution for the initial state is *true*, *i.e.*, $(idle, 0, idle, 0) \in [\![\mathcal{E}]\!](X)$.

There are a few interesting observations that we can make based on this PBES. Firstly, the quantifier that occurs in the μ-calculus formula does not occur in the

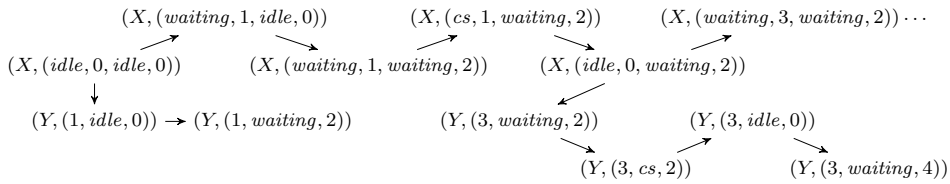


Figure 4.4: Part of an infinite positive proof graph of the bakery example.

PBES above; it has been eliminated with automated techniques [106]. This step simplifies our subsequent analyses. The fact that the variable m can be eliminated implies that the number picked by customer 0 is not relevant, which only becomes apparent after constructing the PBES. It is not obvious how one can draw similar conclusions solely based on the LPS and the formula. Secondly, it is not possible to solve this PBES with traditional instantiation-based techniques, since all positive dependency graphs are infinite. Hence, there is no finite positive proof graph that contains $(X, (idle, 0, idle, 0))$, so even the application of smart heuristics to guide the instantiation does not improve the situation. See Figure 4.4 for a part of a positive proof graph. Lastly, the actual value of n_0 and n_1 is not essential to the problem. What matters is which of the two is larger. This inspired us to investigate symbolic techniques for solving PBESs.

4.3 Reduced Parity Game

In [37], proof graphs were introduced mainly to formalise the concept of witnesses and counterexamples, as implemented in [136]. Instead, we (partially) *solve* PBESs by searching for concise representations of proof graphs. To reason symbolically about the underlying dependency graph of a PBES $\mathcal{E}$, we need to rely on the information contained in $\mathcal{E}$. This is not trivial for PBESs with arbitrary structure [74]. Therefore, we rely on the *clustered recursive form* (CRF) normal form and parity games (which coincide with dependency spaces, see Chapter 3) to simplify the reasoning about the dependencies in a PBES.

In the literature, different approaches to solving PBESs have been proposed. Many of those rely on instantiation of the PBES to a finite Boolean equation system. The BES can then be solved with Gaussian elimination [92] or with a parity game solver [114]. However, for PBESs with an underlying infinite BES, instantiation is not possible. Several symbolic approaches have been proposed to reason about the solution of such a PBES. Most notably, Koolen *et al.* [81] use SMT solvers to find proof graphs and Nagae *et al.* [101, 100] compute reduced proof graphs that finitely represent an infinite proof graph. We extend that latter work to arbitrary PBESs and show how a reduced parity game can be computed, if it is finite, with *PBES quotienting*.

Definition 4.6. Let $G = (V, E, \Omega, \mathcal{P})$ be a parity game for a PBES $\mathcal{E}$. Then $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ is a *reduced parity game*, where:

- $V_r \subseteq 2^V$ is a partition of V,
- $E_r = \{(b, b') \in V_r \times V_r \mid \exists s \in b, t \in b' . s \, E \, t\}$.
- For all $b \in V_r$, $\Omega_r(b) = \Omega(s)$ and $\mathcal{P}_r(b) = \mathcal{P}(s)$ for some $s \in b$.

We say G is the *base game* of G_r.

The intuition behind reduced parity games is that nodes that are in some way equivalent, are grouped. In this way, some infinite parity games can be represented finitely. Note that the priority and player functions (Ω_r and $\mathcal{P}_r$, respectively) that label the blocks are in general not uniquely defined. That is only the case if, for all blocks $b \in V_r$ and nodes $s, t \in b$, $\Omega(s) = \Omega(t)$ and $\mathcal{P}(s) = \mathcal{P}(t)$. We call partitions with that property *consistent*. A reduced game is consistent iff the partition that makes up its blocks is consistent.

We again use bisimulation as equivalence relation on nodes. We remark that bisimulation for parity games relies on the labels that are associated with nodes, contrary to bisimulation for LTSs, which reasons about actions that label transitions (cf. Definition 4.3).

Definition 4.7. Let $G = (V, E, \Omega, \mathcal{P})$ be a parity game for $\mathcal{E}$. A relation $\mathcal{R} \subseteq V \times V$ is a *bisimulation relation* iff for all $s \, \mathcal{R} \, t$:

- $\Omega(s) = \Omega(t)$ and $\mathcal{P}(s) = \mathcal{P}(t)$; and
- If sEs', then there is a node t' such that tEt' and $s' \, \mathcal{R} \, t'$; and
- If tEt', then there is a node s' such that sEs' and $s' \, \mathcal{R} \, t'$.

Nodes s and t are bisimilar, denoted $s \leftrightarroweq t$, iff they are related by some bisimulation relation. Two games G and H are bisimilar iff for every node in G there is a bisimilar node in H and vice versa.

Remark that two nodes (X, v) and (Y, w) may be bisimilar even though they originate from different equations in the PBES, as long as those equations have the same rank and operand. Since bisimilarity is an equivalence relation it induces a partition of the node set V into equivalence classes. This partition is denoted ${}^{V}/_{\leftrightarroweq}$. Note that ${}^{V}/_{\leftrightarroweq}$ is a consistent partition. We call the reduced parity game $G_r = ({}^{V}/_{\leftrightarroweq}, E_r, \Omega_r, \mathcal{P}_r)$, that has G as its base graph (cf. Definition 4.6), the *bisimulation quotient* of G, notation ${}^{G}/_{\leftrightarroweq}$.

Using the CRF normal form and the notions of a (reduced) parity game and bisimulation, we now have a setting similar to Section 4.1. Procedure 2 can thus be applied with only minor changes, see Procedure 3. First, in case we are interested in the solution of a *target node* $\hat{X}(\hat{e})$, where $\hat{e}$ is a ground term, we should provide it as input. The target node plays the same role as the initial state $\hat{d}$ does for an LPS.

Procedure 3: PBES Quotienting

Input: PBES $\mathcal{E}$, **initial partition** π_0, **target node** $\hat{X}(\hat{e})$

1 $n := 0$;
2 **while** π_n *is not stable* **do**
3 $\quad n := n + 1$;
4 $\quad \pi_n := (\pi_{n-1} \setminus \{k\}) \cup \{\mathit{split}(k, k'), \mathit{co\text{-}split}(k, k')\}$ **for some** $k, k' \in \pi_{n-1}$ **such that** $\mathit{split}(k, k')$ and $\mathit{co\text{-}split}(k, k')$ are not equivalent to *false*;
5 $\quad E_n := \{(k, k') \mid \exists (X_i, v) \in \mathsf{sig}(\mathcal{E}).$ $[\![k(X_i, d) \wedge \bigvee_{j \in J_i} (\exists e_j{:}E_j.\, f_j(d, e_j) \wedge k'(X_j, g_j(d, e_j)))]\!]\delta_0[v/d]\}$;
6 $\quad \pi_n := \{k \in \pi_n \mid \hat{k} E_n^* k\}$ **where** $\hat{k} \in \pi_n$ **such that** $[\![\hat{k}(\hat{X}, \hat{e})]\!]$ holds;
7 **return** $^G/_{\leftrightarrows} = (\pi_n, E_n, \Omega, \mathcal{P})$;

Alternatively, if we want to solve the complete PBES, we should skip the reachability check (line 6) in every iteration. Second, we need an adapted definition of the initial partition and the splitting operations.

In the PBES setting, the initial partition needs to distinguish nodes that have a different priority or owner (first bullet of Definition 4.7), so π_0 is set to $\{\{(X, v) \in \mathsf{sig}(\mathcal{E}) \mid \Omega(X) = \Omega(Y) \wedge \mathcal{P}(X) = \mathcal{P}(Y)\} \mid Y \in \mathsf{bnd}(\mathcal{E})\}$; this is the coarsest consistent partition of $\mathsf{sig}(\mathcal{E})$. Since all subsequent partitions π_n are finer than π_0, they are also guaranteed to be consistent. The *split* and *co-split* functions no longer have an argument that defines the action on which the split is based. Below, we use the index i of the predicate variable X_i to construct an expression based on the right-hand side of X_i. Recall that this right-hand side has the shape $\bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge X_j(g_j(d, e_j))$ or $\bigwedge_{j \in J_i} \forall e_j{:}E_j.\, f_j(d, e_j) \Rightarrow X_j(g_j(d, e_j))$, since $\mathcal{E}$ is in CRF.

$$\mathit{split}(k, k') = \bigvee_{X_i \in \mathsf{bnd}(\mathcal{E})} k(X_i, d) \wedge \bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge k'(X_j, g_j(d, e_j))$$

$$\mathit{co\text{-}split}(k, k') = \bigvee_{X_i \in \mathsf{bnd}(\mathcal{E})} k(X_i, d) \wedge \neg \bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge k'(X_j, g_j(d, e_j))$$

Example 4.8. We revisit the bakery protocol example from Section 4.2. This PBES is almost in SRF: the only required change is the addition of the equations for X_{true} and X_{false} to the PBES, the addition of "$\Rightarrow X_{false}$" to the clause on lines 8 and 9 and the addition of the clause $true \Rightarrow X_{true}$ to the equation for Y. Transforming the resulting PBES to CRF and running Procedure 3 on it yields a finite reduced parity game (depicted in Figure 4.5) which contains 14 reachable equivalence classes. Here, we abbreviated state names. For example, in state *wi*, process 0 is waiting and process 1 is idle. Furthermore, in state *ww0*, both processes are waiting, but process 0 has preference to enter the critical section first. States belonging to predicate variable Y are prefixed with *Y-*. Node X_{false} is unreachable from the initial state *ii*, and thus not included in the reduced game that results from Procedure 3. Note the symmetry

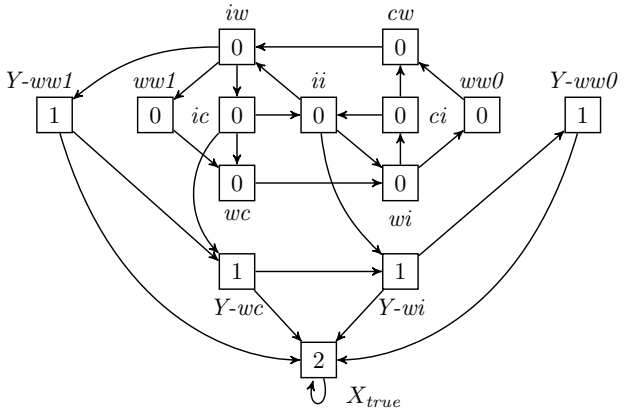


Figure 4.5: Equivalence classes and transitions in the reduced parity game of the bakery protocol example.

between process 0 and process 1 in those states belonging to variable X and also the parallels between X and Y. □

Procedure 3 can be used to solve a PBES as follows. Upon its termination, we obtain the bisimulation quotient of the parity game underlying the PBES $\mathcal{E}$. This quotient is bisimilar to the original game. Since the winner of each node is preserved under bisimilarity [49], solving the quotient—which is itself a parity game—also solves the original game. More formally, given a block $b \in {}^{V}/_{\leftrightarrows}$ and a node $(X, v) \in b$, player $\diamond$ wins b in the quotient ${}^{G}/_{\leftrightarrows}$ if and only if $\diamond$ wins (X, v) in G. By Theorem 3.10, we can also deduce whether $v \in [\![\mathcal{E}]\!](X)$.

Example 4.9. Consider again the reduced parity game of Example 4.8. This reduced parity game is completely won by player $\diamond$. This implies that $\diamond$ also wins all nodes in the original game. We conclude that the solution for the node $(X, (idle, 0, idle, 0))$ is *true*, and that the formula holds, *i.e.*, the bakery protocol does not cause starvation of customers. □

We remark that the procedure presented in this section generalises the procedures presented by Nagae *et al.* in [101] and [100], which only apply to PBESs consisting of predicate formulae that contain no predicate variables within the scope of universal quantifiers.

4.4 Stable Kernel

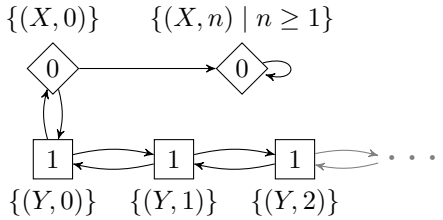
The approach presented in the previous section terminates when the reachable part of the bisimulation quotient is finite and all the operations on data are decidable.

However, we are also interested in solving PBESs for which the bisimulation quotient is not finite. Therefore, we propose an improvement that allows for reasoning about the solution of a single node (X, v), even when some part of the parity game is not finitely representable. This is illustrated by the following example.

Example 4.10. Consider the following PBES:

$$\nu X(n{:}N) = X(n+1) \vee (n = 0 \wedge Y(0))$$
$$\mu Y(n{:}N) = Y(n+1) \wedge (n = 0 \Rightarrow X(0)) \wedge (n > 1 \Rightarrow Y(n-1))$$

The (stable) bisimulation quotient of the parity game of this PBES is infinite and looks as follows:



While this reduced parity game is infinite, the winning strategy in $(X, 0)$ concerns only a finite subgame, namely the subgame that only contains the blocks $\{(X, 0)\}$ and $\{(X, n) \mid n \geq 1\}$. Therefore, to draw conclusions about the solution for $X(0)$, it is not necessary to refine the part of the partition that concerns Y. □

The example suggests that we may in general search for a winning strategy in a—not yet stable—reduced parity game and use that to partially solve a PBES. However, not every strategy obtained that way necessarily induces a proper proof for the original PBES: it is required that the winning strategy witnesses a stable dominion. This is formalised by the concept of a *stable kernel*.

Definition 4.11. Let $G = (V, E, \Omega, \mathcal{P})$ be a parity game of a PBES $\mathcal{E}$ and $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ a consistent reduced parity game, with G as its base game. Furthermore, let $G'_r = (V'_r, E'_r, \Omega'_r, \mathcal{P}'_r)$ be a subgame of G_r. Then, G'_r is a *stable kernel* of G if and only if for all blocks $b, b' \in V'_r$ such that $b\, E'_r\, b'$, b is stable with respect to b'.

The following theorem is the basis for the correctness of Procedure 4, which we will present below.

Theorem 4.12. *Let $G = (V, E, \Omega, \mathcal{P})$ be the parity game for a PBES $\mathcal{E}$ and $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ a consistent reduced parity game with G as its base. Furthermore, let σ_r be a strategy for player $\bigcirc$ in G_r and V'_r a σ_r-dominion in G_r. If $G'_r = G_r \cap V'_r \cap \sigma_r$ is a stable kernel of G, then the base graph of G'_r is a $\bigcirc$-dominion in G.*

Proof. Refer to Figure 4.6 and assume that $G'_r = G_r \cap V'_r \cap \sigma_r$ for some strategy σ_r of player $\bigcirc$ such that V'_r is a σ_r-dominion in G_r.

$$G = (V, E, \Omega, \mathcal{P}) \xrightarrow{\text{base game of}} G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$$
$$\cup\!|$$
$$G'_r = (V'_r, E'_r, \Omega'_r, \mathcal{P}'_r)$$

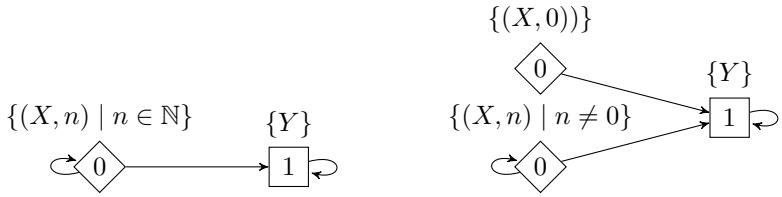
Figure 4.6: Relations between a parity game G, reduced game G_r and reduced subgame G'_r.

Based on σ_r, we define a strategy σ in G such that $\bigcup V'_r$ is a σ-dominion. Let $s \in V$ be some node such that $\mathcal{P}(s) = \bigcirc$ and $s \in b$ for some $b \in V'_r$. Then, $\sigma(s) = s'$ for an arbitrary $s' \in sE \cap \bigcup \sigma_r(b)$. This set is non-empty, since G'_r is a stable kernel of G.

Now we argue that $\bigcup V'_r$ is a σ-dominion in G. Let $(X_0, v_0) \in \bigcup V'_r$ be an arbitrary node and and π some path that starts from (X_0, v_0) and is consistent with σ. Remark that π cannot leave $\bigcup V'_r$ since V'_r is a dominion and G_r over-approximates the transitions of G. The matching path π_r in G_r is $b_0 b_1 \ldots$, where $(X_i, v_i) \in b_i$ for every i. The existence of $b_i \, E_r \, b_{i+1}$ for every i follows from $(X_i, v_i) E (X_{i+1}, v_{i+1})$ and the definition of a reduced game; the equality $\Omega((X_i, v_i)) = \Omega_r(b_i)$ follows from consistency of G_r. We conclude that $\bigcirc$ also wins (X_0, v_0), and that $\bigcup V'_r$ is a σ-dominion in G. □

The following example illustrates that the assumption "G'_r is a stable kernel" from Theorem 4.12 is a necessary condition.

Example 4.13. Consider the PBES $(\nu X(n{:}N) = ((n \neq 0) \wedge X(n)) \vee Y)(\mu Y = Y)$. The figures below depict the initial partition of the parity game of this PBES (on the left-hand side) and the stable partition (on the right-hand side).



The initial partition admits the winning strategy $\sigma(\{(X, n) \mid n \in \mathbb{N}\}) = \{(X, n) \mid n \in \mathbb{N}\}$ for player $\diamond$. However, $\{(X, n) \mid n \in \mathbb{N}\}$ is not stable with respect to itself. As it turns out, there is no winning strategy for $\diamond$ in $\{(X, 0)\}$ in the stable partition. This shows that a strategy that does not induce a stable kernel can in general not be used to draw conclusions about the solution of the PBES under consideration. □

Based on the theory of stable kernels, we propose the following changes to our approach: after every iteration, we solve the parity game that corresponds to the current partition and obtain a winning strategy σ in the block $\hat{k}$, which contains the

Procedure 4: PBES quotienting with stable kernels

Input: PBES $\mathcal{E}$, **initial partition** π_0, **target node** $\hat{X}(\hat{e})$

1 $\rho_0 := \emptyset$;
2 $E_0 := \{(k, k') \in \pi_0 \times \pi_0 \mid \exists (X_i, v) \in \mathsf{sig}(\mathcal{E}).$
$\quad [\![k(X_i, d) \wedge \bigvee_{j \in J_i} \left(\exists e_j{:}E_j.\, f_j(d, e_j) \wedge k'(X_j, g_j(d, e_j))\right)]\!]\delta_0[v/d]\}$;
3 $n := 0$;
4 **while** (π_n, E_n) *is not a stable kernel* **do**
5 $\quad n := n + 1$;
6 $\quad q := (\pi_{n-1} \setminus \{k\}) \cup \{\mathit{split}(k, k'), \mathit{co\text{-}split}(k, k')\}$ **for some** $k, k' \in \pi_{n-1}$
$\quad\quad$ **such that** $\mathit{split}(k, k')$ and $\mathit{co\text{-}split}(k, k')$ are non-empty;
7 $\quad q := q \cup \rho_{n-1}$;
8 $\quad E_n := \{(k, k') \in q \times q \mid \exists (X_i, v) \in \mathsf{sig}(\mathcal{E}).$
$\quad\quad [\![k(X_i, d) \wedge \bigvee_{j \in J_i} \left(\exists e_j{:}E_j.\, f_j(d, e_j) \wedge k'(X_j, g_j(d, e_j))\right)]\!]\delta_0[v/d]\}$;
9 $\quad q := \{k \mid \hat{k} E_n^* k\}$ **where** $(\hat{X}, [\![\hat{e}]\!]) \in \hat{k}$;
10 $\quad (\sigma_\diamond, \sigma_\square) :=$ `solveParityGame(`q, E_n`)`;
11 $\quad \pi_n := \{k \in q \mid \hat{k}\, E_{n,\sigma}^* k\}$ **where** $[\![\hat{k}(\hat{X}, \hat{e})]\!]$ and $\sigma \in \{\sigma_\diamond, \sigma_\square\}$ is winning in $\hat{k}$;
12 $\quad \rho_n := q \setminus \pi_n$;
13 **return** (π_n, E_n);

target node $\hat{X}(\hat{e})$. Then, we construct a minimal σ-dominion containing $\hat{k}$. In the next iteration, we only consider the blocks in that dominion for refinement. When the blocks in the dominion form a stable kernel, the procedure can terminate (by Theorem 4.12). See Procedure 4. We maintain two sets of blocks: π_n contains the blocks in the dominion that we are currently considering and ρ_n contains the other blocks. At line 6, we split a block in π_n and temporarily store the resulting partition in q. Then, the set of blocks of the whole partition, reachable under the new transition relation from the block containing the target node $\hat{X}(\hat{e})$ is computed (lines 7 to 9). Thereby, blocks that are not reachable from the target node are effectively "thrown away", *i.e.*, they are not considered during the next iterations. Since unreachable blocks cannot be part of a minimal dominion for the target node, this does not affect the correctness of the procedure. Subsequently, we solve the parity game consisting of the reachable blocks (line 10, function `solveParityGame`), which can be done with existing algorithms, such as Zielonka's recursive algorithm [140]. As a result, we obtain a strategy for each player; the strategy σ of the player that wins $\hat{k}$ is used to construct a minimal σ-dominion (line 11). We achieve this by following only the transitions of E_n that are consistent with σ, represented here by the relation $E_{n,\sigma}$, starting from $\hat{k}$. The blocks of the dominion are again stored in π_n, the remaining blocks are stored in ρ_n. After every iteration, we check whether π_n is a stable kernel (line 4). If so, the procedure terminates.

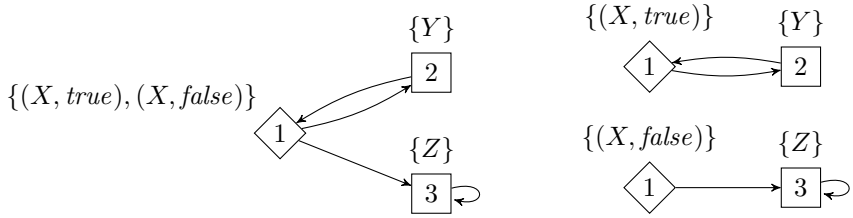
4.5 Stability Under Solution

The procedures presented so far rely purely on bisimulation as the notion of equivalence. However, we are only interested in the solution of the PBES. Bisimulation is much stronger than solution equivalence, resulting in a reduced parity game that is larger than necessary to determine the solution. Besides bisimulation, several other equivalences have been defined in the literature, such as *consistent correlation* [137] and the corresponding preorder *consistent consequence* [33]. This inspired us to investigate how our techniques can benefit from a weaker equivalence relation.

Example 4.14. Consider the following PBES $\mathcal{E}$ with target node $X(true)$:

$$\begin{aligned} \mu X(b{:}B) &= (b \wedge Y) \vee (\neg b \wedge Z) \\ \nu Y &= X(true) \\ \mu Z &= Z \end{aligned}$$

The initial and stable partition of the parity game for $\mathcal{E}$ are respectively:



In the initial partition, the two possible strategies for player $\diamond$ in $\{(X, true), (X, false)\}$ are both losing. Since the transition relation of a reduced parity game is an over-approximation (cf. Definition 4.6, existential quantifier in the second bullet), after splitting the block $\{(X, true), (X, false)\}$, each of the resulting blocks will either have the same or fewer outgoing transitions. Thus, player $\diamond$ has even less capability of winning the resulting blocks $\{(X, true)\}$ and $\{(X, false)\}$, and we conclude that player $\Box$ also wins those blocks. In general, if player $\diamond$ loses a node it owns, removing outgoing edges of that node preserves winning strategies of $\Box$. $\Box$

We formalise this observation in the following definition.

Definition 4.15. Let $\mathcal{E}$ be a PBES and $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ be a consistent reduced parity game. A block $b \in V_r$ is *stable under solution* iff at least one of the following conditions holds:

- b is stable under bisimulation; or
- b is won by $\overline{\mathcal{P}(b)}$, the player who does not own b.

A reduced parity game is stable under solution iff all its blocks are stable under solution.

The intuition behind this is that if player $\Diamond$ loses a block that it owns, then it still loses that block if some outgoing edges are removed, which might occur during splitting. Thus, splitting such a block will not change the solution. Dually, blocks owned by player $\Box$ do not have to be split if $\Diamond$ wins them.

We now proceed by proving that stability under solution is a sufficient condition to preserve the solution. In the proof below, recall that vE denotes the successor set of v under the transition relation E (cf. Definition 2.21).

Theorem 4.16. *Let $\mathcal{E}$ be a PBES, $G = (V, E, \Omega, \mathcal{P})$ its parity game and $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ a consistent reduced parity game that is stable under solution. Then, for all $(X, v) \in \mathsf{sig}(\mathcal{E})$, $v \in [\![\mathcal{E}]\!](X)$ iff player $\Diamond$ wins b in G_r, where $(X, v) \in b$.*

Proof. Let $\mathcal{E}$, G and G_r be as above. Here we provide the proof for the positive case, *i.e.*, we show that if player $\Diamond$ wins b, with $(X, v) \in b$, then $v \in [\![\mathcal{E}]\!](X)$. The proof for the negative case, *viz.* player $\Box$ wins b implies $v \notin [\![\mathcal{E}]\!](X)$, is completely analogous.

We consider an arbitrary (X, v) and b such that $(X, v) \in b$. Let σ_r be a $\Diamond$-strategy that is winning for $\Diamond$ in b and let U be a σ_r-dominion. We show that $\bigcup U$ is a $\Diamond$-dominion in G. Let $s \in V$ and $b' \in U$ be such that $s \in b'$ and $\mathcal{P}(b') = \Diamond$. We define the strategy σ such that $\sigma(s) = s'$ for an arbitrary $s' \in sE \cap \bigcup \sigma_r(b')$. Note that this set is not empty, since b is won by its owner (player $\Diamond$), and, hence, must be stable under bisimulation.

Let $\pi = (X_0, v_0)(X_1, v_1) \ldots$ be an arbitrary path in G that is consistent with σ and that starts from some node $(X_0, v_0) \in \bigcup U$. Remark that π cannot leave $\bigcup U$ since U is a dominion, making it closed for player $\Diamond$ under σ, and G_r over-approximates the transitions of G, giving player $\Box$ less choice in G. The matching path π_r in G_r is $b_0 b_1 \ldots$, where $(X_i, v_i) \in b_i$ for every i. The existence of $b_i \, E_r \, b_{i+1}$ for every i follows from $(X_i, v_i)E(X_{i+1}, v_{i+1})$ and the definition of a reduced game; the equality $\Omega((X_i, v_i)) = \Omega_r(b_i)$ follows from consistency of G_r. We conclude that $\Diamond$ also wins (X, v), and that $\bigcup U$ is a σ-dominion in G. It subsequently follows from Theorem 3.10 that $v \in [\![\mathcal{E}]\!](X)$. $\square$

The idea of stability under solution can be applied as an early termination heuristic. In Procedure 4, the stability check on line 4 can be implemented with solution stability.

4.6 Implementation and Experiments

We implemented the ideas presented in the previous sections in two tools that are part of the mCRL2 toolset [26]. The first tool, `lpssymbolicbisim`, performs minimal model generation on LPSs according to Procedure 2. Upon termination, it produces an LTS. The second tool, called `pbessymbolicbisim`, implements PBES quotienting (cf. Section 4.3) and also the optimisations identified in Sections 4.4 and 4.5.

Figure 4.7 shows an overview of the workflow for both tools in the setting of the mCRL2 toolset. In case we want to perform model checking of μ-calculus formulae,

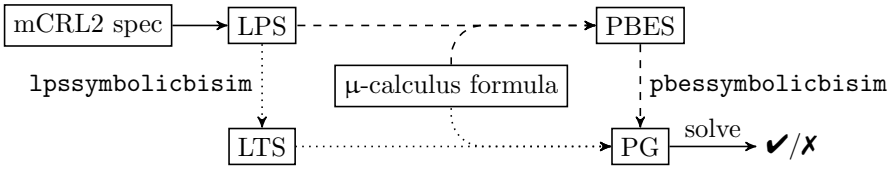


Figure 4.7: Workflow for model checking based on minimal model generation (dotted arrows) and PBES quotienting (dashed arrows).

the mCRL2 specification is first transformed to an LPS. Then, one can choose to use `lpssymbolicbisim` to obtain an LTS, and subsequently use the property to construct a parity game which can be solved. Alternatively, one first constructs a PBES and then applies `pbessymbolicbisim` to obtain the BES.

4.6.1 Implementation

Both `lpssymbolicbisim` and `pbessymbolicbisim` call the Z3 SMT-solver [38] to determine whether one of the sets computed with the functions *split* and *co-split* is empty, *i.e.*, whether its characteristic function is unsatisfiable. When choosing which block to split in each iteration (line 4 of Procedure 2 and line 6 of Procedure 4), preference is given to blocks that are the least far away from the block containing the target node.

A critical component of the implementation is the handling of characteristic functions. To manipulate these expressions, we first of all rely on the mCRL2 term rewrite system. Furthermore, the tools also contain several specialised algorithms that simplify the characteristic functions after splitting a block. Our experience is that it is worthwhile spending some runtime on these simplifications. If the characteristic functions are never minimised, they contain a lot of redundancy and quickly grow prohibitively large. This slows down any subsequent computation by the procedure.

In our implementation, the characteristic functions are simplified in two steps. First, we attempt to eliminate the existential quantifier that originates from the *split* and *co-split* functions. This can be done by distributing the quantifier over other operators, applying the substitution rule for quantifiers, enumerating finite data or applying Fourier-Motzkin elimination [48, 99] to real variables. Details of distributing quantifiers over other operators and the substitution rule can be found in Chapter 7. The second step is to transform the resulting predicate formula to a more compact representation using a combination of *multi-valued decision diagrams* (MDDs) [70] and *multi-terminal decision diagrams* (MTBDDs) [8, 31]. We store all variables of finite data types in the multi-valued nodes and expressions over the remaining variables in the terminals. Although these decision diagrams only eliminate redundancy in variables of a finite data type, this method is effective, since most of our models contain a finite control structure.

4.6.2 Setup

We compare the performance of several approaches: our implementations of minimal model generation and PBES quotienting and the `pbes-cvc4` tool from [81]. We originally also aimed to compare with the tool `PBESSolver` from [100]. However, their implementation has several practical limitations, making a fair comparison impossible. We therefore decided to exclude `PBESSolver` from our experiments. The experiments were performed on a machine with an Intel Core i5 3350P processor and 8 GB of memory running Ubuntu 18.04 and mCRL2 commit hash 066ba9f36b[1] compiled with GCC7.3.

Our set of benchmarks[2] consists of various PBESs that encode different types of decision problems, covering typical linear-time, branching-time and real-time model checking problems, a scheduling problem, recursive functions and behavioural equivalence checking problems. The PBESs encoding model checking problems mostly originate from the set of examples included in mCRL2, which in some cases have been modified to generate infinite state spaces. Classical approaches that generate the state space explicitly fail for all of these models. We remark that most of the models contain multiple concurrent processes. Each model is combined with one or more formal properties in the form of a modal μ-calculus formula to obtain a PBES. More specifically, we verified the following properties:

- two *reachability* properties (the real-time ball game [33]: winning impossible; and the real-time train gate system [12]: action *go(1)* can be executed at time 20);
- two *invariants* (Fischer's real-time mutual exclusion protocol [87] and Lamport's bakery protocol [86]: no deadlock);
- six linear and branching-time properties (the ball game: infinitely often put ball; the train gate: fairness; Fischer's protocol and Lamport's bakery protocol: request must be served; the Concurrent Alternating Bit Protocol (CABP) [82]: a message can be received infinitely often; Hesselink's handshake register [65]: cache consistency, and all writes finish).

The scheduling problem we consider is due to [101]; it encodes a fair trading problem encoded as a PBES. Furthermore, two recursive functions we consider are based on classical benchmarks for verification tools [79]. A modified version of the McCarthy 91 function, as per [100], is represented with the following PBES:

$$\mu M(x, y{:}N) = (x > 10 \wedge x = y + 1) \vee \exists e{:}N.\, x \leq 10 \wedge M(x + 2, e) \wedge M(e, y)$$

Here, $M(x, y)$ is *true* if and only if (x, y) is a solution for the function we represent.

[1] The sources of mCRL2 are available via `https://github.com/mCRL2org/mCRL2`

[2] The experiments are available online via `https://doi.org/10.5281/zenodo.3528141`.

Table 4.1: Runtime comparison between several variants of PBES quotienting and `pbes-cvc4`. All runtimes are in seconds. 't.o.' indicates a time-out and a cross indicates that a PBES cannot be handled.

model	target node/ property	result	PQ			PQ+sk			PQ+sk+ss			`pbes-cvc4`
			$\|V\|$	iter.	time	$\|P\|$	iter.	time	$\|P\|$	iter.	time	time
ball game	winning impossible	*false*	12	65	1.59	11	65	1.73	11	11	0.19	0.27
	inf. often *put_ball*	*true*	2	4	0.01	1	2	<0.01	1	2	<0.01	t.o.
train gate	*go(1)* at time 20	*true*	7	10	1.37	6	8	0.58	6	6	0.46	0.39
	fairness	*false*	15	57	19.58	4	31	9.28	4	31	11.59	✗
Fischer (N=3)	no deadlock	*true*	5	5	0.24	3	4	0.22	3	4	0.24	✗
Fischer (N=4)	request must serve	*false*	18	50	431.43	3	16	9.04	3	3	0.91	✗
bakery	no deadlock	*true*	1	1	<0.01	1	1	<0.01	1	1	<0.01	t.o.
	request must serve	*false*	64	153	6.79	5	17	0.36	5	17	0.37	0.44
Hesselink	cache consistency	*false*		t.o.		20	754	345.89	20	753	348.43	✗
	all writes finish	*false*		t.o.		24	219	11.07	24	259	11.41	✗
CABP	receive inf. often	*true*	79	313	14.12	16	114	2.69	17	111	3.13	✗
trading	inf. run possible	*true*	10	16	0.1	6	9	0.03	6	9	0.04	t.o.
McCarthy	$M(0, 10)$	*true*		t.o.		14	726	45.38	14	725	55.67	✗
	$M(0, 9)$	*false*		t.o.		299	1025	85.67	299	1025	90.04	✗
Takeuchi	$T(3, 2, 1, 3)$	*true*		t.o.		12	54	6.48	12	54	6.62	✗
	$T(3, 2, 1, 2)$	*false*		t.o.		186	79	16.32	185	79	17.92	✗
ABP + buffer	branching bisimilar	*true*	30	55	0.21	29	49	0.24	23	42	0.24	✗

In a similar fashion, we have a PBES for Takeuchi's function [79]:

$$\mu T(x, y, z, w{:}N) = (x \leq y \wedge y = w) \vee (\exists t_1, t_2, t_3{:}N.\, x > y \wedge \\ T(x-1, y, z, t_1) \wedge T(y-1, z, x, t_2) \wedge T(z-1, x, y, t_3) \wedge T(t_1, t_2, t_3, w))$$

Finally, we consider the decision problem whether the Alternating Bit Protocol (ABP) [17] is branching bisimilar to a one-place buffer, both with infinite data. This PBES is encoded using the techniques in [28], as implemented in the mCRL2 tool `lpsbisim2pbes`.

4.6.3 Comparison of PBES solvers

We ran `pbessymbolicbisim` and `pbes-cvc4` for each of the PBESs. The results are listed in Table 4.1. Here, 'PQ' denotes the standard PBES quotienting procedure and '+sk' and '+ss' denote the additional use of the stable kernel and stability under solution, respectively. For each PBES, we report the solution for the target node and the runtime in seconds for each approach. The runtimes reported here do not include the time required to compile the rewriter, which is roughly constant for each PBES. For each PBES quotienting experiment, we also report the size of the resulting reduced parity game or stable kernel, denoted with $|V|$ and $|P|$ respectively, and the number of iterations required to compute it, denoted with 'iter.'. A timeout, set to half an hour, is represented with 't.o.' and we write a cross for the PBESs that cannot be handled.

We observe that the use of stable kernels improves the performance over the basic PBES quotienting procedure for nearly every PBES in our set of benchmarks.

Furthermore, several timeouts occur for 'PQ', while 'PQ+sk' and 'PQ+sk+ss' manage to solve these PBESs. The added value of stability under solution over the stable kernels procedure is not clear. The only instances where it performs significantly better are the ball game with the 'winning impossible' property and Fischer's protocol with the 'request must serve' property. For most other models, stability under solution causes some overhead, leading to a longer runtime.

The runtime of `pbes-cvc4` is very small for the three cases it can solve. However, it fails to provide a solution in most of the cases. Although `pbes-cvc4` and `pbessymbolicbisim` rely on different SMT solvers (CVC4 and Z3, respectively), the differences we observe in our benchmarks can be fully explained by their approach alone. The three cases where a timeout occurs for `pbes-cvc4` (trading, ball game and bakery) are similar: the models contain one or more variables that strictly increase. Since `pbes-cvc4` can only find strategies that induce lasso-shaped plays, it does not terminate for PBESs with infinite winning plays that are not lasso-shaped.

For Fischer and bakery with the no deadlock property and the equivalence problem on ABP and buffer, the stable kernel covers almost the entire reduced dependency space. Only the block containing X_{false} (cf. Section 3.1) is not present in the stable kernel. In those cases, PBES quotienting does not benefit from the optimisation of using stable kernels (Procedure 4).

In order to obtain a result for Takeuchi with the target node $T(3, 2, 1, 2)$, we modified our implementation slightly: after every iteration, we randomly shuffle the order in which blocks are stored. This can affect the *splitting strategy*: the choice of blocks b and b' to be used for splitting. This is discussed in more detail in Section 4.6.5.

4.6.4 Comparison of PBES quotienting and MMG

We also conducted several experiments with our implementation of minimal model generation in the tool `lpssymbolicbisim`. To obtain meaningful results, we made the following changes to our models:

- For the timed models (ball game, train gate and Fischer), we removed the time tags from actions and encoded the timed semantics of mCRL2 using a new process parameter. As a consequence, we cannot verify properties that refer to absolute time. This transformation is implemented in the tool `lpsuntime`.
- We removed the infinite data domain from the Hesselink, CABP, ABP and buffer models.

Without these modifications, MMG cannot compute a bisimulation quotient.

The results are listed in Table 4.2. For MMG, $|P|$ indicates the size of the resulting reduced LTS. For ball game and bakery, we only have to run `lpssymbolicbisim` once (cf. Figure 4.7). On the other hand, to check branching bisimilarity of ABP and the buffer, MMG has to generate the bisimulation quotient for both models before we can compare them with the tool `ltscompare`.

Table 4.2: Runtime comparison between minimal model generation and PBES quotienting. All runtimes are in seconds. 't.o.' indicates a time-out and a cross indicates that a property cannot be handled.

		MMG			PQ+sk+ss		
model	property	$\|P\|$	iter.	time	$\|P\|$	iter.	time
ball game	winning impossible	11	64	0.50	11	11	0.19
	infinitely often *put_ball*				1	2	<0.01
train gate	*go(1)* at time 20		✗		6	6	0.46
	fairness	15	72	15.64	4	31	11.59
Fischer (N=3)	no deadlock	62	180	22.36	3	4	0.24
Fischer (N=4)	request must serve		t.o.		3	3	0.91
bakery	no deadlock	22	52	1.31	1	1	<0.01
	request must serve				5	17	0.37
ABP	branching bisimilar	28	111	5.12	23	42	0.42
buffer		3	2	<0.01			

Minimal model generation cannot be used to model check two of our instances. First, for the train gate model, MMG can compute a bisimulation quotient, but it is not possible to subsequently construct a PBES that accurately encodes the property "*go(1)* at time 20", since the quotient does not contain references to absolute time. Second, MMG times out for the Fischer model with four processes.

For the train gate model with the fairness property, our PBES quotienting performs similarly to MMG. This is partially due to the fact that the state space is partly encoded twice in the PBES, once for each fixpoint in the formula, similar to the bakery PBES of Section 4.2. For most of the other benchmarks, PBES quotienting outperforms MMG.

4.6.5 Splitting Strategy

While performing these experiments, we noticed quite some variability in the results for certain models, especially McCarthy and Takeuchi. Slight alterations in the formulation of the PBES can have a significant effect on the runtime. Closer inspection revealed that the cause is the choice of blocks used to split. In our current implementation, we apply the simple heuristic of giving preference to blocks close to the block containing the target node. The following example shows the importance of the splitting strategy.

Example 4.17. We consider the target node $X(\mathit{true}, 0)$ in the PBES below.

$$\begin{aligned} \nu X(b{:}B, n{:}Z) &= ((b \vee n > 0) \wedge X(b, n-1)) \vee (b \wedge Y) \\ \mu Y &= Y \end{aligned}$$

Here, Z represents the integers. Since splits happen closest to the block containing the target node, the block containing $(X, (\mathit{true}, 0))$ is continuously split with respect to itself, and we obtain the following partition after i iterations (while not performing reachability analysis):

$$\{\{(X,(b,n)) \mid b \vee n \geq i\}, \{(X,(\mathit{false},n)) \mid n \leq 0\}, \{Y\}\} \cup \bigcup_{0<n<i} \{\{(X,(\mathit{false},n))\}\}$$

Splitting $\{(X,(b,n)) \mid b \vee n \geq i\}$ with respect to $\{Y\}$ results in $\{(X,(\mathit{true},n)\}$ and $\{(X,(\mathit{false},n)) \mid n \geq i\}$. This leads to immediate termination, since the former block – which contains the target node – is a stable kernel. In this example, the choice of splitting strategy determines whether PBES quotienting terminates. □

To find a good algorithm or heuristic for this block selection, one can draw inspiration from works on minimal model generation, *e.g.*, [90]. Performing static analysis to obtain invariants (*e.g.* b is invariably *true* for the target node) can be another way to identify which blocks to split. Furthermore, the splitting strategy can be made more robust by introducing some randomness: this is to prevent certain blocks from being ignored indefinitely.

4.7 Related Work

The first works on generating minimal representations from behavioural specifications were written by Bouajjani *et al.* [23]. Later, these ideas were applied to timed automata [2, 124]. Similar to our approach, they rely on bisimulation to compute the minimal quotient directly from a specification. Fisler and Vardi [46] extended this work to include early termination when performing reachability analysis. Our work is similar in spirit to these methods, but it generalises these by allowing to verify properties expressed in the full modal μ-calculus and by supporting infinite-state systems, not limited to real-time systems.

The techniques and theory we present also generalise several other closely related works, such as [101, 100, 81, 75]. Nagae *et al.* [101] transfer the ideas of Bouajjani *et al.* to disjunctive, quantifier-free PBESs and generate finite parity games that can be solved. They later expanded the work to existential PBESs [100]. These fragments of the PBES logic limit the type of properties one can verify. A small set of experimental results shows that their approach is feasible in practice for small academic examples.

Koolen *et al.* [81] use an SMT solver to search for linear proof graphs in disjunctive or conjunctive PBESs. Their technique manages to find solutions for model checking

problems where traditional tools time out. They conclude that even for problems where enumeration of the state space is possible, an instantiation-based approach is not always faster. We remark that the number of unrollings performed by their tool gives a rough indication of the optimal size of the proof graph constructed with our techniques when applied to disjunctive or conjunctive PBESs.

In [75], Keiren *et al.* define two equivalence relations based on bisimulation for BESs. These relations are then used to minimise BESs that represent model checking problems. Experiments show that applying minimisation speeds up the solving procedure, *i.e.*, the time required for minimising and then solving the minimal BES is lower than the time required to solve the original BES. Whereas [75] applies explicit-state techniques by working directly on a BES, our work is based on a symbolic representation. The disadvantage of the explicit approach of [75] is that it requires one to instantiate a PBES to BES first, which can be time consuming. Furthermore, the instantiation does not terminate for infinite-state systems.

Fontana *et al.* [47] construct symbolic proof trees to check alternation-free μ-calculus formulae on timed automata. To recursively prove (sub)formulas, they unfold the transition relation according to a set of proof rules they propose. This approach allows a larger class of properties than UPPAAL [12], which only supports a subset of TCTL. Contrary to our approach, the proof they produce is not necessarily minimal with respect to bisimulation.

Tripakis and Yovine [124] also identified the problem that the characteristic functions should be as compact as possible in order to improve the scalability (cf. Section 4.6). They develop a specialised partition-refinement technique for the setting of timed automata such that the characteristic functions are always conjunctive, *i.e.*, they represent a convex set of nodes. Preserving convexity comes at a cost, however: the resulting stable partition can be finer that the bisimulation quotient.

Although our work was not inspired by counterexample-guided abstraction refinement (CEGAR) [29], we see many similarities. CEGAR works by taking an abstract version of the model under consideration and feeding it to a model checker. If a counterexample is found, then this is compared to the original model to determine whether the counterexample is spurious. If so, the abstraction is refined to exclude the spurious trace. Then the procedure is repeated with the refined abstraction. The algorithm terminates when the model checker returns *true* or a valid counterexample is found. Our procedure that finds stable kernels essentially refines with respect to 'spurious dominions'. Compared to our approach, CEGAR typically supports a less expressive class of properties, such as ACTL or LTL.

The recent work by Kobayashi *et al.* [80] shows how to check formulas from the logic Mu-Arithmetic on recursive single-threaded programs. They achieve this by encoding the problem in a *hierarchical equation system* (HES), a logic which practically coincides with PBES, except that it only allows integers as data type. A semi-decision procedure solves HESs by translating the equations to *constrained Horn clauses* (CHC) and feeding them to a CHC solver.

4.8 Conclusion

We presented an approach to solving arbitrarily-structured PBESs with infinite data, which enables solving of a larger set of PBESs than possible with existing tools. This improves the state-of-the-art for model checking and equivalence checking on (concurrent) systems with infinite data.

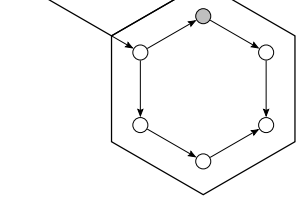
A drawback of performing quotienting on the level of PBESs is that this process has to be repeated for each property that needs to be checked (cf. Figure 4.7). On the other hand, for minimal model generation, the LTS needs to be generated only once, after which multiple properties can be checked by constructing multiple BESs, which is a relatively cheap operation. However, as we have shown, PBES quotienting also has several fundamental advantages, which improve its applicability in a practical setting.

Further study is required to fully understand the effects of splitting strategy (cf. Section 4.6.5). We believe that a good strategy, perhaps based on heuristics, can significantly improve the scalability of our approach.

When checking fairness properties, the state space is typically encoded twice in the corresponding PBES. The PBES from Section 4.2 is a perfect example. In the current implementation, the same work is sometimes done twice for different predicate variables. The procedures can be further optimised by exploiting this symmetry.

Another possible direction is to further weaken the equivalence relation on dependency graph nodes. Here, one can draw inspiration from equivalence relations defined on parity games, for instance as defined in [36].

Chapter 5



The Inconsistent Labelling Problem

As we have seen in the introduction, one of the main challenges in model checking is the state-space explosion problem. The many interleavings of concurrent processes can cause the state space to grow exponentially with the number of components. This is particularly problematic if there is little interaction between the processes and they mainly contain independent behaviour. *Partial-order reduction* (POR) attempts to tackle this problem by recognising independent behaviour, identifying similar interleavings and (hopefully) exploring only one interleaving from each equivalence class. The crux is that POR is typically applied *on-the-fly*, *i.e.*, while generating the state space. This avoids the scenario where the full state-space is first generated—potentially running out of memory in the process—and then reduced.

Literature sees many variants of POR; the main ones are *ample sets* [111], *persistent sets* [52] and *stubborn sets* [127, 132]. The most basic approaches apply relatively weak conditions, thus achieving great reductions. However, the only property they preserve is the absence/presence of deadlocks. Stronger conditions are necessary to preserve reachability properties or logics such as LTL or CTL. For each of the variants listed above, sufficient conditions for preservation of stutter-trace equivalence have been identified. Since LTL without the next operator (LTL_{-X}) is invariant under finite stuttering, this allows one to check most LTL properties under POR.

However, the correctness proofs for these methods are intricate and not reproduced

often. For stubborn sets, LTL$_{-X}$-preserving conditions and an accompanying correctness result were first presented in [126], and discussed in more detail in [128]. While trying to reproduce the proof for [128, Theorem 2] (see also Theorem 5.9 in this chapter), we ran into an issue while trying to prove a certain property of the construction used in the original proof [128, Construction 1]. This led us to discover that stutter-trace equivalence is not necessarily preserved. We will refer to this as the *inconsistent labelling problem*. The essence of the problem is that POR in general, and the proofs in [128] in particular, reason mostly about actions, which label the transitions. The only relevance of the state labelling is that it determines which actions are *visible*. On the other hand, stutter-trace equivalence and the LTL semantics are purely based on state labels. The correctness proof in [128] does not deal properly with this disparity. Further investigation shows that the same problem also occurs in two works of Beneš *et al.* [14, 15], who apply ample sets to state/event LTL model checking.

Consequently, any application of stubborn sets in LTL$_{-X}$ model checking is possibly unsound, both for safety and liveness properties. The correctness of several theories in the literature [85, 91, 129] relies on the incorrect theorem. In this chapter, we investigate what the impact of the inconsistent labelling problem is. First, we prove the existence of the inconsistent labelling problem with a counter-example to correctness of the LTL$_{-X}$-preserving conditions (Section 5.2). This counter-example is valid for weak stubborn sets and, with a small modification, in a non-deterministic setting for strong stubborn sets. Then, we propose to strengthen one of the stubborn set conditions (Section 5.3) and show that this modification resolves the issue (Theorem 5.12). Finally, we analyse in which circumstances the inconsistent labelling problem occurs (Section 5.4). This includes a thorough analysis of Petri nets and several different notions of invisible transitions and atomic propositions (Section 5.5). Based on our analysis, we discuss the impact on related work (Section 5.6).

Our investigation shows that probably all practical implementations of stubborn sets compute an approximation which resolves the inconsistent labelling problem. Furthermore, POR methods based on the standard independence relation, such as ample sets and persistent sets, are not affected.

5.1 Preliminaries

Most POR methods assume the existence of both state labels and transition labels, *i.e.*, actions. Therefore, the transition systems in this chapter differ slightly from the rest of the thesis: we will use *labelled state transition systems*, instead of LTSs (Definition 2.2). In addition to a finite set Act of action labels, we assume the existence of some fixed set of atomic propositions AP to label the states.

Definition 5.1. A *labelled state transition system*, short *LSTS*, is a directed graph $TS = (S, \rightarrow, \hat{s}, L)$, where:

- S is the state space;

- $\rightarrow \subseteq S \times Act \times S$ is the transition relation;
- $\hat{s} \in S$ is the initial state; and
- $L : S \rightarrow 2^{AP}$ is a function that labels states with atomic propositions.

On top of the notation that we already had for LTSs, we introduce the following notions. Given a path $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \ldots$, the *trace* of π is the sequence of state labels observed along π, *viz.* $L(s_0)L(s_1)L(s_2)\ldots$. A set $\mathcal{I} \subseteq Act$ of invisible actions is chosen such that if (but not necessarily only if) $a \in \mathcal{I}$, then for all states s and t, $s \xrightarrow{a} t$ implies $L(s) = L(t)$. Note that this definition allows the set $\mathcal{I}$ to be under-approximated; this is useful in practice, since it is not always trivial to determine exactly which actions never change the state labelling. An action that is not invisible is called *visible*.

5.1.1 Stubborn sets

In POR, *reduction functions* play a central role. A reduction function $r : S \rightarrow 2^{Act}$ indicates which transitions to explore in each state. When starting at the initial state $\hat{s}$, a reduction function induces a *reduced LSTS* as follows.

Definition 5.2. Let $TS = (S, \rightarrow, \hat{s}, L)$ be an LSTS and $r : S \rightarrow 2^{Act}$ a reduction function. Then the *reduced LSTS* induced by r is defined as $TS_r = (S_r, \rightarrow_r, \hat{s}, L_r)$, where L_r is the restriction of L on S_r, and S_r and $\rightarrow_r$ are the smallest sets such that the following holds:

- $\hat{s} \in S_r$; and
- if $s \in S_r$, $s \xrightarrow{a} t$ and $a \in r(s)$, then $t \in S_r$ and $s \xrightarrow{a}_r t$.

Note that we have $\rightarrow_r \subseteq \rightarrow$. In the remainder of this chapter, we assume the reduced LSTS is finite. This is essential for the correctness of the approach detailed below. In general, a reduction function is not guaranteed to preserve almost any property of an LSTS. Below, we list a number of conditions that have been proposed in the literature; they aim to preserve LTL_{-X}. The notion of a *key action*, referred to in the conditions, is defined as follows: an action a is a key action in s iff for all paths $s \xrightarrow{a_1 \ldots a_n} s'$ such that $a_1 \notin r(s), \ldots, a_n \notin r(s)$, it holds that $s' \xrightarrow{a}$. Note that a key action must be enabled in s: by setting $n = 0$, we have $s = s'$ and $s \xrightarrow{a}$. We typically denote key actions by a_{key}.

D0 If $\mathit{enabled}(s) \neq \emptyset$, then $r(s) \cap \mathit{enabled}(s) \neq \emptyset$.

D1 For all $a \in r(s)$ and $a_1 \notin r(s), \ldots, a_n \notin r(s)$, if $s \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are states $s', s'_1, \ldots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'_n$.

D2 Every enabled action in $r(s)$ is a key action in s.

D2w If $\mathit{enabled}(s) \neq \emptyset$, then $r(s)$ contains a key action in s.

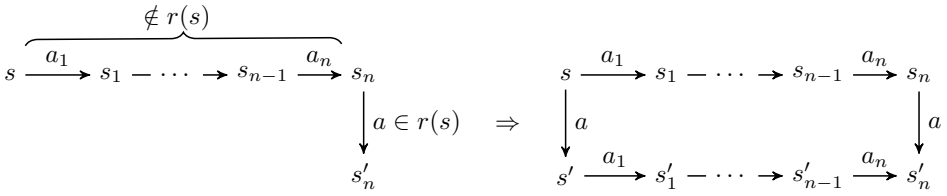


Figure 5.1: Visual representation of condition **D1**.

V If $r(s)$ contains an enabled visible action, then $r(s)$ contains all visible actions.

I If an invisible action is enabled in s, then $r(s)$ contains an invisible key action.

L For every visible action a, every cycle in the reduced LSTS contains a state s such that $a \in r(s)$.

These conditions are used to define *strong* and *weak* stubborn sets in the following way.

Definition 5.3. A reduction function $r : S \to 2^{Act}$ is a *strong stubborn set* iff for all states $s \in S$, the conditions **D0**, **D1**, **D2**, **V**, **I**, **L** all hold.

Definition 5.4. A reduction function $r : S \to 2^{Act}$ is a *weak stubborn set* iff for all states $s \in S$, the conditions **D1**, **D2w**, **V**, **I**, **L** all hold.

Below, we also use 'weak/strong stubborn set' to refer to the set of actions $r(s)$ in some state s. A stubborn set can never introduce new deadlocks in the reduced LSTS, either by **D0** or **D2w**. Condition **D1** enforces that a key action $a_{\mathsf{key}} \in r(s)$ does not disable other paths that are not selected for the stubborn set. A visual representation of condition **D1** can be found in Figure 5.1. When combined, **D1** and **D2w** are sufficient conditions for preservation of deadlocks. Condition **V** enforces that the paths $s \xrightarrow{a_1 \ldots a_n a} s'_n$ and $s \xrightarrow{a a_1 \ldots a_n} s'_n$ in **D1** contain the same sequence of visible actions. The purpose of condition **I** is to preserve the possibility to perform an invisible action, if one is enabled. Finally, we have condition **L** to deal with the *action-ignoring problem*, which occurs when an action is never selected for the stubborn set and always ignored. Since we assume that the reduced LSTS is finite, it suffices to reason in **L** about every cycle instead of every infinite path. The combination of **V**, **I** and **L** helps to preserve divergences (infinite paths containing only invisible actions).

Example 5.5. Consider the LSTS in Figure 5.2. The dashed states and transitions are present in the original LSTS, but not in the reduced version. Grey states are labelled with $\{q\}$, other states do not have any atomic proposition assigned as label. Other LSTSs in the current chapter are visualised in a similar way.

Actions d and e are visible; actions a, b and c may be invisible. In this case, we choose the set of invisible actions to be maximal, *i.e.*, $\mathcal{I} = \{a, b, c\}$. In the initial

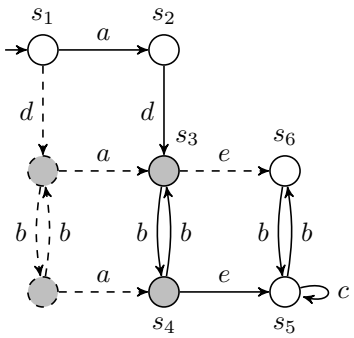


Figure 5.2: Example of an LSTS that is reduced under POR.

state s_1, we have $r(s_1) = \{a\}$. Remark that a is a key action in s_1, since for all prefixes π of db^ω, we have $s_1 \xrightarrow{\pi a}$.

In states s_3 and s_4 we must have $b \in r(s_3)$, respectively $b \in r(s_4)$, by condition **I**. Condition **L** can be satisfied in the cycle consisting of s_3 and s_4 by either setting $e \in r(s_3)$ or $e \in r(s_4)$; here we have opted for the latter. Actually, $e \in r(s_4)$ is also enforced by **D1**, since we have $s_4 \xrightarrow{ecb} s_6$ and $s_4 \xrightarrow{ec} s_5$, but not $s_4 \xrightarrow{bec} s_6$ or $s_4 \xrightarrow{ce} s_5$. Consequently, $\{b\}$ and $\{b, c\}$ are not stubborn sets in s_4. □

Conditions **D0** and **D2** together imply **D2w**, and thus every strong stubborn set is also a weak stubborn set. Since the reverse does not necessarily hold, weak stubborn sets might offer more reduction.

5.1.2 Weak and Stutter Equivalence

To reason about the similarity of an LSTS TS and its reduced LSTS TS_r, we introduce the notions *weak equivalence*, which operates on actions, and *stutter equivalence*, which operates on states. The definitions are generic, so that they can also be used in Section 5.5.

Definition 5.6. Two paths π and π' are weakly equivalent with respect to a set of actions A, notation $\pi \sim_A \pi'$, if and only if they are both finite or both infinite and their respective projections on $Act \setminus A$ are equal.

Definition 5.7. The *no-stutter trace* under labelling L of a path $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$ is the sequence of those $L(s_i)$ such that $i = 0$ or $L(s_i) \neq L(s_{i-1})$. Paths π and π' are stutter equivalent under L, notation $\pi \triangleq_L \pi'$, iff they are both finite or both infinite, and they yield the same no-stutter trace under L.

We typically consider weak equivalence with respect to the set of invisible actions $\mathcal{I}$. In that case, we write $\pi \sim \pi'$. We also omit the subscript for stutter equivalence when reasoning about the standard labelling function and write $\pi \triangleq \pi'$. Note that stutter

equivalence is invariant under finite repetitions of state labels, hence its name. We lift both equivalences to LSTSs, and say that TS and TS' are *weak-trace equivalent* iff for every initial path π in TS, there is a weakly equivalent initial path π' in TS' and vice versa. Likewise, TS and TS' are *stutter-trace equivalent* iff for every initial path π in TS, there is a stutter equivalent initial path π' in TS' and vice versa.

In general, weak equivalence and stutter equivalence are incomparable, even for initial paths. However, for some LSTSs, these notions *are* related in a certain way. We formalise this in the following definition.

Definition 5.8. An LSTS is *labelled consistently* iff for all initial paths π and π', $\pi \sim \pi'$ implies $\pi \triangleq \pi'$.

It follows from the definition that, if an LSTS TS is labelled consistently and weak-trace equivalent to a subgraph TS', then TS and TS' are also stutter-trace equivalent.

Stubborn sets as defined in the previous section aim to preserve stutter-trace equivalence between the original and the reduced LSTS. The motivation behind this is that two stutter-trace equivalent LSTSs satisfy exactly the same formulae [9] in LTL_{-X}. The following theorem, which is frequently cited in the literature [85, 91, 129], aims to show that stubborn sets indeed preserve stutter-trace equivalence. Its original formulation reasons about the validity of an arbitrary LTL_{-X} formula. Here, we give the alternative formulation based on stutter-trace equivalence.

Theorem 5.9. *[128, Theorem 2] Given an LSTS TS and a weak/strong stubborn set r, then the reduced LSTS TS_r is stutter-trace equivalent to TS.*

The original proof correctly concludes that the stubborn set method preserves the order of visible actions in the reduced LSTS, *i.e.*, $TS \sim TS_r$. However, this only implies preservation of stutter-trace equivalence ($TS \triangleq TS_r$) if the full LSTS is labelled consistently, so Theorem 5.9 is invalid in the general case. In the next section, we will see a counter-example which exploits this fact.

5.2 Counter-Example

Consider the LSTS in Figure 5.3, which we will refer to as TS^C. There is only one atomic proposition q, which holds in the grey states and is false in the other states. The initial state $\hat{s}$ is marked with an incoming arrow. First, note that this LSTS is deterministic. The actions a_1, a_2 and a_3 are visible and a and a_{key} are invisible. By setting $r(\hat{s}) = \{a, a_{\mathsf{key}}\}$, which is a weak stubborn set, we obtain a reduced LSTS TS_r^C that does not contain the dashed states and transitions. The original LSTS contains the trace $\emptyset\{q\}\emptyset\emptyset\{q\}^\omega$, obtained by following the path with actions $a_1a_2aa_3^\omega$. However, the reduced LSTS does not contain a stutter equivalent trace. This is also witnessed by the LTL_{-X} formula $\Box(q \Rightarrow \Box(q \vee \Box\neg q))$. An equivalent μ-calculus formula is $\nu X.\,([\top]X \wedge Q \Rightarrow (\nu Y.\,[\top]Y \wedge (Q \vee (\nu Z.\,[\top]Z \wedge \neg Q))))$, if the semantics is such that $[\![Q]\!]\eta\delta = \{s \mid q \in L(s)\}$. These formulas hold for TS_r^C, but not for TS^C.

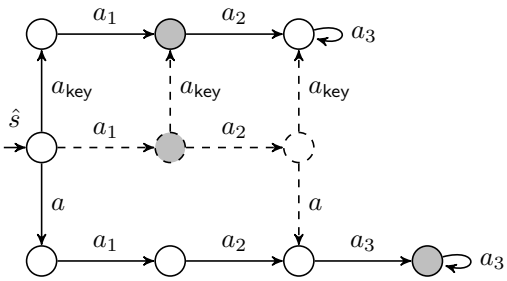


Figure 5.3: Counter-example showing that stubborn sets do not preserve stutter-trace equivalence. Grey states are labelled with $\{q\}$. The dashed transitions and states are not present in the reduced LSTS.

A very similar example can be used to show that strong stubborn sets suffer from the same problem. Consider again the LSTS in Figure 5.3, but assume that $a = a_{\mathsf{key}}$, making the LSTS no longer deterministic. Now, $r(\hat{s}) = \{a\}$ is a strong stubborn set and again the trace $\emptyset\{q\}\emptyset\emptyset\{q\}^\omega$ is not preserved in the reduced LSTS. In Section 5.3.3, we will see why the inconsistent labelling problem does not occur for deterministic systems under strong stubborn sets.

The core of the problem lies in the fact that condition **D1**, even when combined with **V**, does not enforce that the two paths it considers are stutter equivalent. Consider the paths $s \xrightarrow{a}$ and $s \xrightarrow{a_1 a_2 a}$ and assume that $a \in r(s)$ and $a_1 \notin r(s), a_2 \notin r(s)$. Condition **V** ensures that at least one of the following two holds: (i) a is invisible, or (ii) a_1 and a_2 are invisible. Half of the possible scenarios are depicted in Figure 5.4; the other half are symmetric. Again, the grey states (and only those states) are labelled with $\{q\}$.

The two cases delimited with a solid line are problematic. In both LSTSs, the paths $s \xrightarrow{a_1 a_2 a} s'$ and $s \xrightarrow{a a_1 a_2} s'$ are weakly equivalent, since a is invisible. However, they are not stutter equivalent, and therefore these LSTSs are not labelled consistently. The topmost of these two LSTSs forms the core of the counter-example TS^C, with the rest of TS^C serving to satisfy condition **D2**/**D2w**.

5.3 Strengthening Condition D1

To fix the issue with inconsistent labelling, we propose to strengthen condition **D1** as follows.

D1' For all $a \in r(s)$ and $a_1 \notin r(s), \ldots, a_n \notin r(s)$, if $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are states $s', s'_1, \ldots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'_n$. Furthermore, if a is invisible, then $s_i \xrightarrow{a} s'_i$ for every $1 \leq i < n$.

This new condition **D1'** provides a form of *local* consistent labelling when one of $a_1, \ldots, a_n$ is visible. In this case, **V** implies that a is invisible and, consequently, the

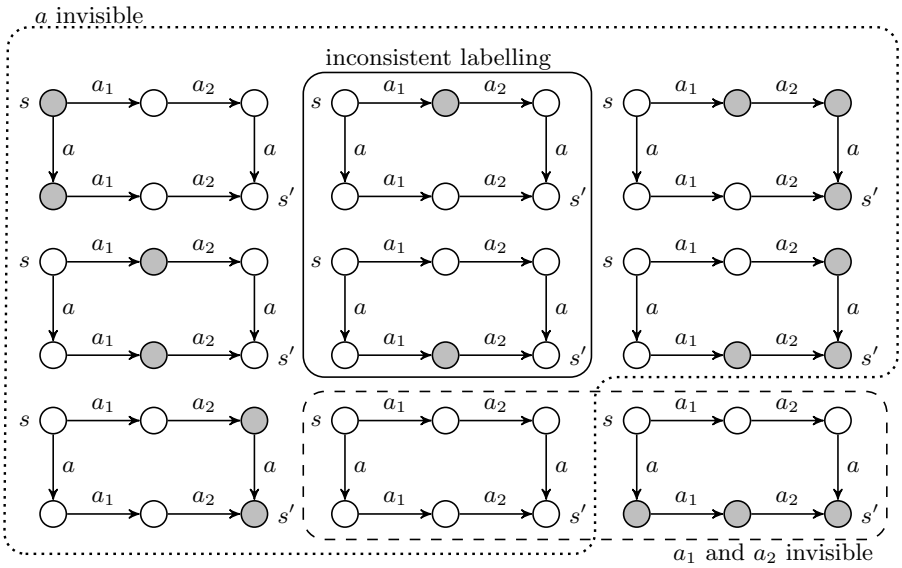


Figure 5.4: Nine possible scenarios when $a \in r(s)$ and $a_1 \notin r(s), a_2 \notin r(s)$, according to conditions **D1** and **V**. The dotted and dashed lines indicate when a or a_1, a_2 are invisible, respectively.

presence of transitions $s_i \xrightarrow{a} s_i'$ implies $L(s_i) = L(s_i')$. Hence, the problematic cases of Figure 5.4 are resolved; a correctness proof is given below.

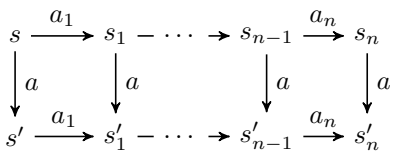
Condition **D1'** is very similar to condition **C1** [50], which is common in the context of ample sets. However, **C1** requires that action a is *globally* independent of each of the actions $a_1, \ldots, a_n$, while **D1'** merely requires a kind of *local* independence. Persistent sets [52] also rely on a condition similar to **D1'**, and require local independence.

5.3.1 Implementation

In practice, most, if not all, implementations of stubborn sets approximate **D1** based on a binary relation $\rightsquigarrow_s$ on actions. This relation may (partly) depend on the current state s and it is defined such that **D1** can be satisfied by ensuring that if $a \in r(s)$ and $a \rightsquigarrow_s a'$, then also $a' \in r(s)$. A set satisfying **D0**, **D1**, **D2**, **V** and **I** or **D1**, **D2w**, **V** and **I** can be found by searching for a suitable *strongly connected component* in the graph $(Act, \rightsquigarrow_s)$. Condition **L** is dealt with by other techniques.

Practical implementations construct $\rightsquigarrow_s$ by analysing how any two actions a and a' interact. If a is enabled, the simplest (but not necessarily the best possible) strategy is to make $a \rightsquigarrow_s a'$ if and only if a and a' access at least one variable in common. This can be relaxed, for instance, by not considering commutative accesses, such as writing to and reading from a FIFO buffer. As a result, $\rightsquigarrow_s$ can only detect reduction

opportunities in (sub)graphs of the shape



where $a \in r(s)$ and $a_1 \notin r(s), \ldots, a_n \notin r(s)$. The presence of the vertical a transitions in $s_1, \ldots, s_{n-1}$ implies that **D1'** is also satisfied by such implementations. More details on a possible implementation of stubborn sets can be found in Chapter 6, where we apply POR to parity games.

5.3.2 Correctness

To show that **D1'** indeed resolves the inconsistent labelling problem, we reproduce the construction in the original proof [128, Construction 1] in two lemmata and show that it preserves stutter equivalence. Below, recall that $\rightarrow_r$ indicates which transitions occur in the reduced state space.

Lemma 5.10. *Let r be a weak stubborn set, where condition **D1** is replaced by **D1'**, and $\pi = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$ a path such that $a_1 \notin r(s_0), \ldots, a_n \notin r(s_0)$ and $a \in r(s_0)$. Then, there is a path $\pi' = s_0 \xrightarrow{a}_r s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$ such that $\pi \triangleq \pi'$.*

Proof. The existence of π' follows directly from condition **D1'**. Due to condition **V** and our assumption that $a_1 \notin r(s_0), \ldots, a_n \notin r(s_0)$, it cannot be the case that a is visible and at least one of $a_1, \ldots, a_n$ is visible. If a is invisible, then the traces of $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$ and $s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$ are equivalent, since **D1'** implies that $s_i \xrightarrow{a} s'_i$ for every $0 \leq i \leq n$, so $L(s'_i) = L(s_i)$. Otherwise, all of $a_1, \ldots, a_n$ are invisible, so the sequences of labels observed along π and π' have the shape $L(s_0)^{n+1}L(s'_0)$ and $L(s_0)L(s'_0)^{n+1}$, respectively. We conclude that $\pi \triangleq \pi'$. □

Lemma 5.11. *Let r be a weak stubborn set, where condition **D1** is replaced by **D1'**, and $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$ a path such that $a_i \notin r(s_0)$ for any a_i that occurs in π. Then, the following holds:*

- *If π is of finite length $n > 0$, there exist an action a_{key}, a state s'_n such that $s_n \xrightarrow{a_{\mathsf{key}}} s'_n$ and a path $\pi' = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$.*
- *If π is infinite, there exists a path $\pi' = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \ldots$ for some action a_{key}.*

In either case, $\pi \triangleq \pi'$.

Proof. Let K be the set of key actions in $r(s)$. If a_1 is invisible, K contains at least one invisible action, due to **I**. Otherwise, a_1 is visible, and we reason that K is not

empty (condition **D2w**) and all enabled actions in $r(s_0)$, and thus also all actions in K, are invisible, due to **V**. In the remainder, let a_{key} be an invisible key action.

In case π has finite length n, the existence of $s_n \xrightarrow{a_{\mathsf{key}}} s'_n$ and $s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$ follows from the definition of key actions and **D1'**, respectively.

If π is infinite, we use a reasoning similar to that of Kőnig's Lemma [77]. We can apply the definition of key actions and **D1'** successively to obtain a path $\pi_i = s_0 \xrightarrow{a_{\mathsf{key}}} s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} s'_i$ for every $i \geq 0$, with $s_j \xrightarrow{a_{\mathsf{key}}} s'_j$ for every $1 \leq j < i$. Since the reduced state space is finite, infinitely many of these paths must use the same state as s'_0. At most one of them ends at s'_0 (the one with $i = 0$), so infinitely many continue from s'_0. Of them, infinitely many must use the same s'_1, again because the reduced state space is finite. Again, at most one of them is lost because of ending at s'_1. This reasoning can continue without limit, proving the existence of $\pi' = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \ldots$, with $s_j \xrightarrow{a_{\mathsf{key}}} s'_j$ for every $j \geq 0$.

Since a_{key} is invisible, we have $L(s_j) = L(s'_j)$ for every $j \geq 0$. This implies $\pi \triangleq \pi'$. □

Lemmata 5.10 and 5.11 coincide with branches 1 and 2 of [128, Construction 1], respectively, but contain the stronger result that $\pi \triangleq \pi'$. Thus, when applied in the proof of [128, Theorem 2] (see also Theorem 5.9), this yields the result that stubborn sets with condition **D1'** preserve stutter-trace equivalence.

Theorem 5.12. *Given an LSTS TS and weak/strong stubborn set r, where condition **D1** is replaced by **D1'**, then the reduced LSTS TS_r is stutter-trace equivalent to TS.*

We do not reproduce the complete proof here, but provide insight into the application of the lemmata with the following example. A complete proof for the setting of parity games is provided in Chapter 6.

Example 5.13. Consider the path obtained by following $a_1a_2a_3$ in Figure 5.5. Lemmata 5.10 and 5.11 show that $a_1a_2a_3$ can always be mimicked in the reduced LSTS, while preserving stutter equivalence. In this case, the path is mimicked by the path corresponding to $a_{\mathsf{key}}a_2a_1a'_{\mathsf{key}}a_3$, drawn with dashes. The new path reorders the actions a_1, a_2 and a_3 according to the construction of Lemma 5.10 and introduces the key actions a_{key} and a'_{key} according to Lemma 5.11. □

We remark that Lemma 5.11 also holds if the reduced LSTS is infinite, but finitely branching.

5.3.3 Deterministic LSTSs

As already noted in Section 5.2, strong stubborn sets for deterministic systems do not suffer from the inconsistent labelling problem. The following lemma, which also appeared as [131, Lemma 4.2], shows why.

Lemma 5.14. *For deterministic LSTSs, conditions **D1** and **D2** together imply **D1'**.*

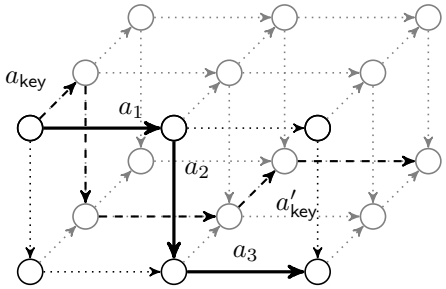


Figure 5.5: Example of how the trace a_1, a_2, a_3 can be mimicked by introducing additional actions and moving a_2 to the front (dashed trace). Transitions that are drawn in parallel have the same label.

Proof. Let TS be a deterministic LSTS, $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$ a path in TS and r a reduction function that satisfies **D1** and **D2**. Furthermore, assume that $a \in r(s_0)$ and $a_1 \notin r(s_0), \ldots, a_n \notin r(s_0)$. By applying **D1**, we obtain the path $\pi' = s_0 \xrightarrow{a} s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$, which satisfies the first part of condition **D1'**. With **D2**, we have $s_i \xrightarrow{a} s'_i$ for every $1 \leq i \leq n$. Then, we can also apply **D1** to every path $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_i} s_i \xrightarrow{a} s'_i$ to obtain, for all $1 \leq i \leq n$, paths $\pi_i = s_0 \xrightarrow{a} s^i_0 \xrightarrow{a_1} s^i_1 \xrightarrow{a_2} \cdots \xrightarrow{a_i} s'_i$. Since TS is deterministic, every path π_i must coincide with a prefix of π'. We conclude that $s'_i = s'_i$ and so the requirement that $s_i \xrightarrow{a} s'_i$ for every $1 \leq i \leq n$ is also satisfied. □

5.4 Safe Logics

In this section, we will identify two logics, *viz.* reachability and CTL_{-X}, which are not affected by the inconsistent labelling problem. This is either due to their limited expressivity or the extra POR conditions that are required.

5.4.1 Reachability properties

Although the counter-example of Section 5.2 shows that stutter-trace equivalence is in general not preserved by stubborn sets, some fragments of LTL_{-X} are preserved. One such class of properties is reachability properties, which are of the shape $\Box f$ or $\Diamond f$, where f is a formula not containing temporal operators.

Theorem 5.15. *Let TS be an LSTS, r a reduction function that satisfies either **D0**, **D1**, **D2**, **V** and **L** or **D1**, **D2w**, **V** and **L** and TS_r the reduced LSTS. For all possible labellings $l \subseteq AP$, TS contains a path to a state s such that $L(s) = l$ iff TS_r contains a path to a state s' such that $L(s') = l$.*

Proof. The 'if' case is trivial, since TS_r is a subgraph of TS. For the 'only if' case, we reason as follows. Let $TS = (S, \rightarrow, \hat{s}, L)$ be an LSTS and $\pi = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n$

a path such that $s_0 = \hat{s}$. We mimic this path by repeatedly taking some enabled action a that is in the stubborn set, according to the following schema. Below, we assume the path to be mimicked contains at least one visible action. Otherwise, its first state would have the same labelling as s_n.

1. If there is an i such that $a_i \in r(s_0)$, we consider the smallest such i, *i.e.*, $a_1 \notin r(s_0), \ldots, a_{i-1} \notin r(s_0)$. Then, we can shift a_i forward by **D1**, move towards s_n along $s_0 \xrightarrow{a_i} s'_0$ and continue by mimicking $s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_{i-1}} s_i \xrightarrow{a_{i+1}} \cdots \xrightarrow{a_n} s_n$.
2. If all of $a_1 \notin r(s_0), \ldots, a_n \notin r(s_0)$, then, by **D0** and **D2** or by **D2w**, there is a key action a_{key} in s_0. By the definition of key actions and **D1**, a_{key} leads to a state s'_0 from which we can continue mimicking the path $s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'_n$. Note that $L(s_n) = L(s'_n)$, since a_{key} is invisible by condition **V**.

The second case cannot be repeated infinitely often, due to condition **L**. Hence, after a finite number of steps, we reach a state s'_n with $L(s'_n) = L(s_n)$. □

We remark that more efficient mechanisms for reachability checking under POR have been proposed, such as condition **S** [132], which can replace **L**, or conditions based on *up-sets* [121]. Another observation is that model checking of LTL$_{-X}$ properties can be reduced to reachability checking by computing the cross-product of a Büchi automaton and an LSTS [9], in the process resolving the inconsistent labelling problem. Peled [112] shows how this approach can be combined with POR, but please note the correctness issues detailed in [122].

5.4.2 Deterministic LSTSs and CTL$_{-X}$ Model Checking

In this section, we consider the inconsistent labelling problem in the setting of CTL$_{-X}$ model checking. When applying stubborn sets in that context, stronger conditions are required to preserve the branching structure that CTL$_{-X}$ reasons about. Namely, the original LSTS must be deterministic and one more condition needs to be added [50]:

C4 Either $r(s) = Act$ or $r(s) \cap \mathit{enabled}(s) = \{a\}$ for some $a \in Act$.

We slightly changed its original formulation to match the setting of stubborn sets. A weaker condition, called **Ä8**, which does not require determinism of the whole LSTS is proposed in [130]. With **C4**, strong and weak stubborn sets collapse, as shown by the following lemma.

Lemma 5.16. *Conditions **D2w** and **C4** together imply **D0** and **D2**.*

Proof. Let *TS* be an LSTS, s a state and r a reduction function that satisfies **D2w** and **C4**. Condition **D0** is trivially implied by **C4**. Using **C4**, we distinguish two cases: either $r(s)$ contains precisely one enabled action a, or $r(s) = Act$. In the former case, this single action a must be a key action, according to **D2w**. Hence, **D2**, which requires that all enabled actions in $r(s)$ are key actions, is satisfied. Otherwise, if $r(s) = Act$, we consider an arbitrary action a that satisfies **D2**'s precondition that

$s \xrightarrow{a}$. Given a path $s \xrightarrow{a_1 \ldots a_n}$, the condition that $a_1 \notin r(s), \ldots, a_n \notin r(s)$ only holds if $n = 0$. We conclude that **D2**'s condition $s \xrightarrow{a_1 \ldots a_n a}$ is satisfied by the assumption $s \xrightarrow{a}$. □

It follows from Lemmata 5.14 and 5.16 and Theorem 5.12 that CTL$_{-X}$ model checking of deterministic systems with stubborn sets does not suffer from the inconsistent labelling problem. The same holds for condition **Ä8**, as already shown in [130].

5.5 Petri Nets

In this section, we discuss the impact of the inconsistent labelling problem on *Petri nets*. Petri nets are a widely-known formalism for modelling concurrent processes and have seen frequent use in the application of stubborn-set theory [20, 91, 132, 133]. A Petri net contains a set of *places* P and a set of *structural transitions* T. *Arcs* between places and structural transitions are weighted according to a total function $W : (P \times T) \cup (T \times P) \to \mathbb{N}$. The state space of the underlying LSTS is the set $\mathcal{M}$ of all *markings*; a marking m is a function $P \to \mathbb{N}$, which assigns a number of *tokens* to each place. The LSTS contains a transition $m \xrightarrow{t} m'$ iff $m(p) \geq W(p,t)$ and $m'(p) = m(p) - W(p,t) + W(t,p)$ for all places $p \in P$. As before, we assume that the LSTS contains some labelling function $L : \mathcal{M} \to 2^{AP}$. More details on the labels are given below. In this section, markings and structural transitions take over the role of states and actions respectively. The set of markings reachable under $\to$ from some *initial marking* $\hat{m}$ is denoted $\mathcal{M}_{reach}$. Note that the LSTS of a Petri Net is deterministic. We want to stress that all the theory in this section is specific for the semantics defined above.

Example 5.17. Consider the Petri net with initial marking $\hat{m}$ on left of Figure 5.6. Here, all arcs are weighted 1, except for the arc from p_5 to t_3, which is weighted 2. Its LSTS is infinite, but the substructure reachable from $\hat{m}$ is depicted on the right. The number of tokens in each of the places $p_1, \ldots, p_6$ is inscribed in the nodes, the state labels (if any) are written beside the nodes.

The LSTS practically coincides with the counter-example of Section 5.2. Only the self-loops are missing and the state labelling, with atomic propositions q, q_p and q_l, differs slightly; the latter will be explained later. For now, note that t and t_{key} are invisible and that the trace $\emptyset\{q_p\}\emptyset\emptyset\{q\}$, which occurs when firing transitions $t_1 t_2 t t_3$ from $\hat{m}$, can be lost when reducing with weak stubborn sets. □

In the remainder of this section, we fix a Petri net $(P, T, W, \hat{m})$ and its LSTS $(\mathcal{M}, \to, \hat{m}, L)$. Below, we consider three different types of atomic propositions. Firstly, polynomial propositions [20] are of the shape $f(p_1, \ldots, p_n) \bowtie k$ where f is a polynomial over $p_1, \ldots, p_n$, $\bowtie \in \{<, \leq, >, \geq, =, \neq\}$ and $k \in \mathbb{Z}$. Such a proposition holds in a marking m iff $f(m(p_1), \ldots, m(p_n)) \bowtie k$. A linear proposition [91] is similar, but the function f over places must be linear and $f(0, \ldots, 0) = 0$, *i.e.*, linear propositions

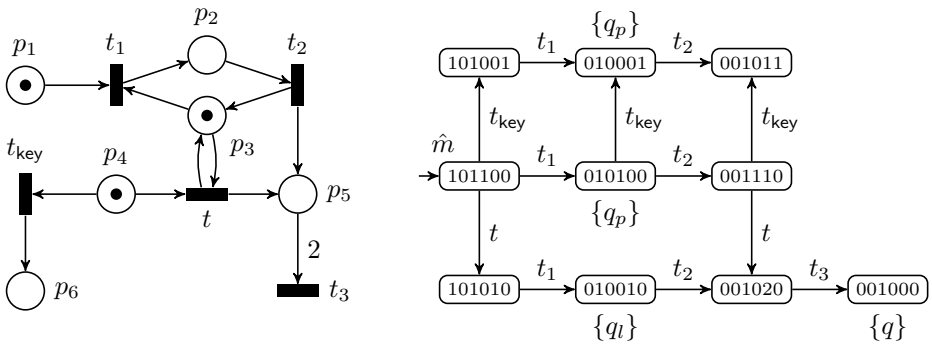


Figure 5.6: Example of a Petri net whose LSTS suffers from the inconsistent labelling problem.

are of the shape $k_1 p_1 + \cdots + k_n p_n \bowtie k$, where $k_1, \ldots, k_n, k \in \mathbb{Z}$. Finally, we have arbitrary propositions [133], whose shape is not restricted and which can hold in any given set of markings.

Several other types of atomic propositions can be encoded as polynomial propositions. For example, *fireable*(t) [20, 91], which holds in a marking m iff t is enabled in m, can be encoded as $\prod_{p \in P} \prod_{i=0}^{W(p,t)-1} (p - i) \geq 1$. The proposition *deadlock*, which holds in markings where no structural transition is enabled, does not require special treatment in the context of POR, since it is already preserved by **D1** and **D2w**. The sets containing all linear and polynomial propositions are henceforward called AP_l and AP_p, respectively. The corresponding labelling functions are defined as $L_l(m) = L(m) \cap AP_l$ and $L_p(m) = L(m) \cap AP_p$ for all markings m. Below, the two stutter equivalences $\triangleq_{L_l}$ and $\triangleq_{L_p}$ that follow from the new labelling functions are abbreviated $\triangleq_l$ and $\triangleq_p$, respectively. Note that $AP \supseteq AP_p \supseteq AP_l$ and $\triangleq \subseteq \triangleq_p \subseteq \triangleq_l$.

For the purpose of introducing several variants of invisibility, we reformulate and generalise the definition of invisibility from Section 5.1. Given an atomic proposition $q \in AP$, a relation $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$ is *q-invisible* if and only if $(m, m') \in \mathcal{R}$ implies $q \in L(m) \Leftrightarrow q \in L(m')$. We consider a structural transition t q-invisible iff its corresponding relation $\{(m, m') \mid m \xrightarrow{t} m'\}$ is q-invisible. Invisibility is also lifted to sets of atomic propositions: given a set $AP' \subseteq AP$, relation $\mathcal{R}$ is *AP'-invisible* iff it is q-invisible for all $q \in AP'$. If $\mathcal{R}$ is AP-invisible, we plainly say that $\mathcal{R}$ is *invisible*. AP'-invisibility and invisibility carry over to structural transitions. We sometimes refer to invisibility as *ordinary invisibility* for emphasis. Note that the set of invisible structural transitions $\mathcal{I}$ is no longer an under-approximation, but contains exactly those structural transitions t for which $m \xrightarrow{t} m'$ implies $L(m) = L(m')$ (cf. Section 5.1).

We are now ready to introduce three orthogonal variations on invisibility.

Definition 5.18. Let $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{M}$ be a relation on markings. Then,

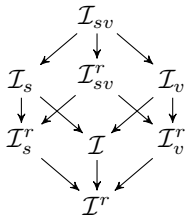


Figure 5.7: Lattice of sets of invisible actions. Arrows represent a subset relation.

- $\mathcal{R}$ is *reach q-invisible* [132] iff $\mathcal{R} \cap (\mathcal{M}_{reach} \times \mathcal{M}_{reach})$ is q-invisible; and
- $\mathcal{R}$ is *value q-invisible* iff
  - $q = (f(p_1, \ldots, p_n) \bowtie k)$ is polynomial and for all $(m, m') \in \mathcal{R}$, we have that $f(m(p_1), \ldots, m(p_n)) = f(m'(p_1), \ldots, m'(p_n))$; or
  - q is not polynomial and $\mathcal{R}$ is q-invisible.

Intuitively, under reach q-invisibility, all pairs of reachable markings $(m, m') \in \mathcal{R}$ have to agree on the labelling of q. For value invisibility, the value of the polynomial f must never change between two markings $(m, m') \in \mathcal{R}$. Reach and value invisibility are lifted to structural transitions and sets of atomic propositions as before, *i.e.*, by taking $\mathcal{R} = \{(m, m') \mid m \xrightarrow{t} m'\}$ when considering invisibility of t.

Definition 5.19. A structural transition t is *strongly q-invisible* iff the set $\{(m, m') \mid \forall p \in P : m'(p) = m(p) + W(t, p) - W(p, t)\}$ is q-invisible.

Strong invisibility does not take the presence of a transition $m \xrightarrow{t} m'$ into account, and purely reasons about the effects of t. Value invisibility and strong invisibility are new in the current work, although strong invisibility was inspired by the notion of invisibility that is proposed by Varpaaniemi in [133]. Our definition of strong invisibility weakens the conditions of Varpaaniemi.

We indicate the sets of all value, reach and strongly invisible structural transitions with $\mathcal{I}_v$, $\mathcal{I}^r$ and $\mathcal{I}_s$ respectively. Since $\mathcal{I}_v \subseteq \mathcal{I}$, $\mathcal{I}_s \subseteq \mathcal{I}$ and $\mathcal{I} \subseteq \mathcal{I}^r$, the set of all their possible combinations forms the lattice shown in Figure 5.7. In the remainder, the weak equivalence relations that follow from each of the eight invisibility notions are abbreviated, *e.g.*, $\sim_{\mathcal{I}^r_{sv}}$ becomes $\sim^r_{sv}$.

Example 5.20. Consider again the Petri net and LSTS from Example 5.17. We can define q_l and q_p as linear and polynomial propositions, respectively:

- $q_l := p_3 + p_4 + p_6 = 0$ is a linear proposition, which holds when neither p_3, p_4 nor p_6 contains a token. Structural transition t is q_l-invisible, because $m \xrightarrow{t} m'$ implies that $m(p_3) = m'(p_3) \geq 1$, and thus neither m nor m is labelled with q_l. On the other hand, t is not value q_l-invisible (by the transition $101100 \xrightarrow{t} 101010$) or strongly reach q_l-invisible (by 010100 and 010010).

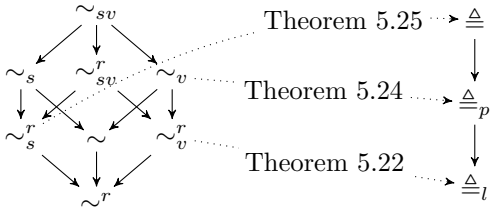


Figure 5.8: Two lattices containing variations of weak equivalence and stutter equivalence, respectively. Solid arrows indicate a subset relation inside the lattice; dotted arrows follow from the indicated theorems and show when the LSTS of a Petri net is labelled consistently.

However, t_{key} is strongly value q_l-invisible: it moves a token from p_4 to p_6 and hence never changes the value of $p_3 + p_4 + p_6$.

- $q_p := (1 - p_3)(1 - p_5) = 1$ is a polynomial proposition, which holds in all reachable markings m where $m(p_3) = m(p_5) = 0$ or $m(p_3) = m(p_5) = 2$. Structural transition t is reach value q_p-invisible, but not q_p-invisible (by $002120 \xrightarrow{t} 002030$) or strongly reach q_p invisible. Strong value q_p-invisibility of t_{key} follows immediately from the fact that the adjacent places of t_{key}, *viz.* p_4 and p_6, do not occur in the definition of q_p.

This yields the state labelling which is shown in Example 5.17. □

Given a weak equivalence relation $R_\sim$ and a stutter equivalence relation $R_{\triangleq}$, we write $R_\sim \preceq R_{\triangleq}$ to indicate that $R_\sim$ and $R_{\triangleq}$ yield consistent labelling. We spend the rest of this section investigating under which notions of invisibility and propositions from the literature, the LSTS of a Petri net is labelled consistently. More formally, we check for each weak equivalence relation $R_\sim$ and each stutter equivalence relation $R_{\triangleq}$ whether $R_\sim \preceq R_{\triangleq}$. This tells us when existing stubborn set theory can be applied without problems. The two lattices containing all weak and stuttering equivalence relations are depicted in Figure 5.8; each dotted arrow represents a consistent labelling result. Before we continue, we first introduce an auxiliary lemma.

Lemma 5.21. *Let I be a set of invisible structural transitions and L some labelling function. If for all $t \in I$ and paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots$ and $\pi' = m_0 \xrightarrow{t} m_0' \xrightarrow{t_1} m_1' \xrightarrow{t_2} \dots$, it holds that $\pi \triangleq_L \pi'$, then $\sim_I \preceq \triangleq_L$.*

Proof. We assume that the following holds for all paths and $t \in I$:

$$m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \cdots \triangleq_L m_0 \xrightarrow{t} m_0' \xrightarrow{t_1} m_1' \xrightarrow{t_2} \dots \qquad (\dagger)$$

We consider two initial paths π and π' such that $\pi \sim_I \pi'$ and prove that $\pi \triangleq_L \pi'$. The proof proceeds by induction on the combined number of invisible structural transitions (taken from I) in π and π'. In the base case, π and π' contain only visible

structural transitions, and $\pi \sim_I \pi$' implies $\pi = \pi'$ since Petri nets are deterministic. Hence, $\pi \triangleq_L \pi'$.

For the induction step, we take as hypothesis that, for all initial paths π and π' that together contain at most k invisible structural transitions, $\pi \sim_I \pi'$ implies $\pi \triangleq_L \pi'$. Let π and π' be two arbitrary initial paths such that $\pi \sim_I \pi'$ and the total number of invisible structural transitions contained in π and π' is k. We consider the case where an invisible structural transition is introduced in π', the other case is symmetric. Let $\pi' = \sigma_1\sigma_2$ for some σ_1 and σ_2. Let $t \in I$ be some invisible structural transition and $\pi'' = \sigma_1 t \sigma_2'$ such that σ_2 and σ_2' contain the same sequence of structural transitions. Clearly, we have $\pi' \sim_I \pi''$. Here, we can apply our original assumption (†), to conclude that $\sigma_2 \triangleq t\sigma_2'$, *i.e.*, the extra stuttering step t thus does not affect the labelling of the remainder of π''. Hence, we have $\pi' \triangleq_L \pi''$ and, with the induction hypothesis, $\pi \triangleq_L \pi''$. Note that π and π'' together contain $k+1$ invisible structural transitions.

In case π and π' together contain an infinite number of invisible structural transitions, $\pi \sim_I \pi'$ implies $\pi \triangleq_L \pi'$ follows from the fact that the same holds for all finite prefixes of π and π' that are related by $\sim_I$. □

The following theorems each focus on a class of atomic propositions and show which notion of invisibility is required for the LSTS of a Petri net to be labelled consistently. In the proofs, we use a function d_t, defined as $d_t(p) = W(t,p) - W(p,t)$ for all places p, which indicates how structural transition t changes the state. Furthermore, we also consider functions of type $P \to \mathbb{N}$ as vectors of type $\mathbb{N}^{|P|}$. This allows us to compute the pairwise addition of a marking m with d_t ($m + d_t$) and to indicate that t does not change the marking ($d_t = 0$).

Theorem 5.22. *Under reach value invisibility, the LSTS underlying a Petri net is labelled consistently for linear propositions,* i.e., $\sim_v^r \preceq \triangleq_l$.

Proof. Let $t \in \mathcal{I}_v^r$ be a reach value invisible structural transition such that there exist reachable markings m and m' with $m \xrightarrow{t} m'$. If such a t does not exist, then $\sim_v^r$ is the reflexive relation and $\sim_v^r \preceq \triangleq_l$ is trivially satisfied. Otherwise, let $q := f(p_1, \ldots, p_n) \bowtie k$ be a linear proposition. Since t is reach value invisible and f is linear, we have $f(m) = f(m') = f(m + d_t) = f(m) + f(d_t)$ and thus $f(d_t) = 0$. It follows that, given two paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots$ and $\pi' = m_0 \xrightarrow{t} m_0' \xrightarrow{t_1} m_1' \xrightarrow{t_2} \ldots$, the addition of t does not influence f, since $f(m_i) = f(m_i) + f(d_t) = f(m_i + d_t) = f(m_i')$ for all i. As a consequence, t also does not influence q. With Lemma 5.21, we deduce that $\sim_v^r \preceq \triangleq_l$. □

Whereas in the linear case one can easily conclude that π and π' are stutter equivalent under f, in the polynomial case, we need to show that f is constant under all value invisible structural transitions t, even in markings where t is not enabled. This follows from the following proposition.

Proposition 5.23. *Let $f : \mathbb{N}^n \to \mathbb{Z}$ be a polynomial function, $a, b \in \mathbb{N}^n$ two constant vectors and $c = a - b$ the difference between a and b. Assume that for all $x \in \mathbb{N}^n$*

such that $x \geq b$, where $\geq$ denotes pointwise comparison, it holds that $f(x) = f(x+c)$. Then, f is constant in the vector c, i.e., *$f(x) = f(x+c)$ for all $x \in \mathbb{N}^n$.*

Proof. Let f, a, b and c be as above and let $\mathbf{1} \in \mathbb{N}^n$ be the vector containing only ones. Given some arbitrary $x \in \mathbb{N}^n$, consider the function $g_x(t) = f(x+t\cdot\mathbf{1}+c) - f(x+t\cdot\mathbf{1})$. For sufficiently large t, it holds that $x + t \cdot \mathbf{1} \geq b$, and it follows that $g_x(t) = 0$ for all sufficiently large t. This can only be the case if g_x is the zero polynomial, *i.e.*, $g_x(t) = 0$ for all t. As a special case, we conclude that $g_x(0) = f(x+c) - f(x) = 0$. □

The intuition behind this is that $f(x + c) - f(x)$ behaves like the directional derivative of f with respect to c. If the derivative is equal to zero in infinitely many x, f must be constant in the direction of c. We will apply this result in the following theorem.

Theorem 5.24. *Under value invisibility, the LSTS underlying a Petri net is labelled consistently for polynomial propositions,* i.e., *$\sim_v \preceq \triangleq_p$.*

Proof. Let $t \in \mathcal{I}_v$ be a value invisible structural transition, m and m' two markings with $m \xrightarrow{t} m'$, and $q := f(p_1, \ldots, p_n) \bowtie k$ a polynomial proposition. Note that infinitely many such (not necessarily reachable) markings exist in $\mathcal{M}$, so we can apply Proposition 5.23 to obtain $f(m) = f(m + d_t)$ for all markings m. It follows that, given two paths $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots$ and $\pi' = m_0 \xrightarrow{t} m_0' \xrightarrow{t_1} m_1' \xrightarrow{t_2} \ldots$, the addition of t does not alter the value of f, since $f(m_i) = f(m_i + d_t) = f(m_i')$ for all i. As a consequence, t also does not change the labelling of q. Application of Lemma 5.21 yields $\sim_v \preceq \triangleq_p$. □

Varpaaniemi shows that the LSTS of a Petri net is labelled consistently for arbitrary propositions under his notion of invisibility [133, Lemma 9]. Our notion of strong visibility, and especially strong reach invisibility, is weaker than Varpaaniemi's invisibility, so we generalise the result to $\sim_s^r \preceq \triangleq$.

Theorem 5.25. *Under strong reach visibility, the LSTS underlying a Petri net is labelled consistently for arbitrary propositions,* i.e., *$\sim_s^r \preceq \triangleq$.*

Proof. Let $t \in \mathcal{I}_s^r$ be a strongly reach invisible structural transition and $\pi = m_0 \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots$ and $\pi' = m_0 \xrightarrow{t} m_0' \xrightarrow{t_1} m_1' \xrightarrow{t_2} \ldots$ two paths. Since, $m_i' = m_i + d_t$ for all i, it holds that either (i) $d_t = 0$ and $m_i = m_i'$ for all i; or (ii) each pair (m_i, m_i') is contained in $\{(m, m') \mid \forall p \in P : m'(p) = m(p) + W(t,p) - W(p,t)\}$, which is the set that underlies strong reach invisibility of t. In both cases, $L(m_i) = L(m_i')$ for all i. It follows from Lemma 5.21 that $\sim_s^r \preceq \triangleq$. □

To show that the results of the above theorems cannot be strengthened, we provide two negative results.

Theorem 5.26. *Under ordinary invisibility, the LSTS underlying a Petri net is not necessarily labelled consistently for arbitrary propositions,* i.e., *$\sim \not\preceq \triangleq$.*

Proof. Consider the Petri net from Example 5.17 with the arbitrary proposition q_l. Disregard q_p for the moment. Structural transition t is q_l-invisible, hence the paths corresponding to $t_1t_2tt_3$ and $tt_1t_2t_3$ are weakly equivalent under ordinary invisibility. However, they are not stutter equivalent. □

Theorem 5.27. *Under reach value invisibility, the LSTS underlying a Petri net is not necessarily labelled consistently for polynomial propositions,* i.e., $\sim_v^r \not\preceq \triangleq_p$.

Proof. Consider the Petri net from Example 5.17 with the polynomial proposition $q_p := (1 - p_3)(1 - p_5) = 1$ from Example 5.20. Disregard q_l in this reasoning. Structural transition t is reach value q_p-invisible, hence the paths corresponding to $t_1t_2tt_3$ and $tt_1t_2t_3$ are weakly equivalent under reach value invisibility. However, they are not stutter equivalent for polynomial propositions. □

It follows from Theorems 5.26 and 5.27 and transitivity of $\subseteq$ that Theorems 5.22, 5.24 and 5.25 cannot be strengthened further. In terms of Figure 5.8, this means that the dotted arrows cannot be moved downward in the lattice of weak equivalences and cannot be moved upward in the lattice of stutter equivalences. The implications of these findings on related work will be discussed in the next section.

5.6 Related Work

There are many works in the literature that apply stubborn sets. We will consider several works that aim to preserve LTL_{-X} and discuss whether they are correct when it comes to the inconsistent labelling problem. Furthermore, we also identify several unrelated issues.

Liebke and Wolf [91] present an approach for efficient CTL model checking on Petri nets. For some formulas, they can reduce CTL model checking to LTL model checking, which allows greater reductions under POR. They rely on the incorrect LTL preservation theorem, and since they apply the techniques on Petri nets with ordinary invisibility, their theory is incorrect (Theorem 5.26). Similarly, the overview of stubborn set theory presented by Valmari and Hansen in [132] applies reach invisibility and does not necessarily preserve LTL_{-X}. Varpaaniemi [133] also applies stubborn sets to Petri nets, but relies on a visibility notion that is stronger than strong invisibility. The correctness of these results is thus not affected (Theorem 5.25).

A generic implementation of weak stubborn sets is proposed by Laarman *et al.* [85]. They use abstract concepts such as guards and transition groups to implement POR in a way that is agnostic of the input language. The theory they present includes condition **D1**, which is too weak and thus incorrect, but the accompanying implementation follows the framework of Section 5.3.1, and thus it is correct by Theorem 5.12. The implementations proposed in [132, 139] are similar, albeit specific for Petri nets.

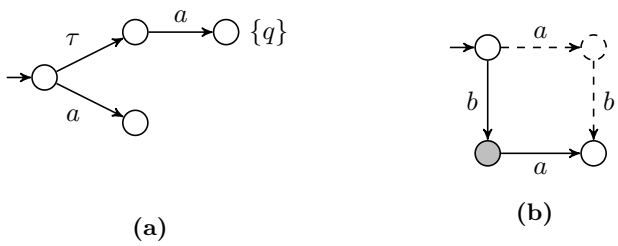


Figure 5.9: Counter-examples for theories in two related works.

Others [51, 62] perform action-based model checking and thus strive to preserve weak trace equivalence or inclusion. As such, they do not suffer from the problems discussed here, which applies only to state labels.

Although Beneš *et al.* [14, 15] rely on ample sets, and not on stubborn sets, they also discuss weak trace equivalence and stutter-trace equivalence. In fact, they present an equivalence relation for traces that is a combination of weak and stutter equivalence. The paper includes a theorem that weak equivalence implies their new state/event equivalence [14, Theorem 6.5]. However, the counter-example in Figure 5.9a shows that this consistent labelling theorem does not hold. Here, the action τ is invisible, and the two paths in this transition system are thus weakly equivalent. However, they are not stutter equivalent, which is a special case of state/event equivalence. Although the main POR correctness result [14, Corollary 6.6] builds on the incorrect consistent labelling theorem, its correctness does not appear to be affected. An alternative proof can be constructed based on Lemmas 5.10 and 5.11.

Bønneland *et al.* [20] apply stubborn-set based POR to two-player Petri nets, and their reachability semantics expressed as a *reachability game*. Since their approach only concerns reachability, it is not affected by the inconsistent labelling problem (see Section 5.4). Unfortunately, their POR theory is nevertheless unsound, contrary to what is claimed in [20, Theorem 17]. In reachability games, player 1 tries to reach one of the *goal* states, while player 2 tries to avoid them. Bønneland *et al.* propose a condition **R** that guarantees that all goal states in the full game are also reachable in the reduced game. However, the reverse is not guaranteed: paths that do not contain a goal state are not necessarily preserved, essentially endowing player 1 with more power. Consider the (solitaire) reachability game depicted in Figure 5.9b, in which all edges belong to player 2 and the only goal state is indicated with grey. Player 2 wins the non-reduced game by avoiding the goal state via the edges labelled with a and then b. However, $\{b\}$ is a stubborn set—according to the conditions of [20]—in the initial state, and the dashed transitions are thus eliminated in the reduced game. Hence, player 2 is forced to move the token to the goal state and player 1 wins in the reduced game. In the mean time, the authors of [20] confirmed and resolved the issue in [21].

The current work is not the first to point out mistakes in POR theory. In [122],

Siegel presents a flaw in an algorithm that combines POR and on-the-fly model checking [112]. In that setting, POR is applied on the product of an LSTS and a Büchi automaton. We briefly sketch the issue here. Let q be a state of the LSTS and s a state of the Büchi automaton. While investigating a transition $(q, s) \xrightarrow{a} (q', s')$, condition **C3**, which—like condition **L**—aims to solve the action ignoring problem, incorrectly sets $r(q, s') = \mathit{enabled}(q)$ instead of $r(q, s) = \mathit{enabled}(q)$.

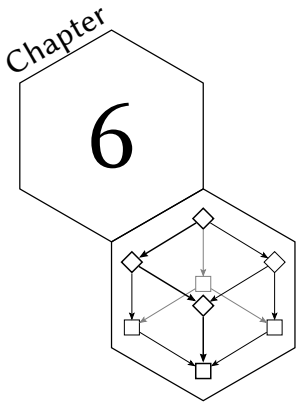
5.7 Conclusion

We discussed the inconsistent labelling problem for preservation of stutter-trace equivalence with stubborn sets. The issue is relatively easy to repair by strengthening condition **D1**. For Petri nets, altering the definition of invisibility can also resolve inconsistent labelling depending on the type of atomic propositions. The impact on applications presented in related works seems to be limited: the problem is typically mitigated in the implementation, since it is very hard to compute **D1** exactly. This is also a possible explanation for why the inconsistent labelling problem has not been noticed for so many years.

Since this is not the first error found in POR theory [122], a more rigorous approach to proving its correctness, *e.g.* using proof assistants, would provide more confidence.

Chapter 6

Partial-Order Reduction for Parity Games



As we have seen in the previous chapter, partial-order reduction (POR) is a popular technique for combating the state explosion problem. However, a major drawback of POR is that most variants at best preserve only a fragment of a given logic, such as LTL or CTL* without the next operator (LTL_{-X}/CTL^*_{-X}) [50] or the weak modal μ-calculus [119]. Furthermore, the variants of POR that preserve a branching time logic impose significant restrictions on the reduction by only allowing the prioritisation of exactly one action at a time. This decreases the amount of reduction achieved.

In this chapter, we address these shortcomings by applying POR to parity games. In the context of model checking, parity games suffer from the same state-space explosion that labelled transition systems do. Exploring the state space of a parity game under POR can be a very effective way to address this. We extend the definition of a parity game to include edge labels, such that it can be interpreted as an LSTS (Definition 5.1). This allows us to use the conditions from Chapter 5, which ensure that the reduction function used to reduce the parity game is correct, *i.e.*, preserves the winning player of the parity game (Theorem 6.8). Furthermore, we identify improvements for the reduction by investigating the typical structure of a parity game that encodes a model checking question. Our POR technique can be used to speed up the solving of PBESs, which are a high-level representation of parity games (see Chapter 3). We thus address the open question of how to apply POR

to PBESs [73, 114, 115, 138]. Since PBESs induce labelled parity games that are not necessarily deterministic, we extend the implementation framework of [85] with support for non-determinism. The ideas are implemented in an experimental tool that solves PBESs.

Our approach has two distinct benefits over traditional POR techniques that operate on transition systems. First, it is the first work that enables the use of partial-order reduction for model checking of the full modal μ-calculus. Second, the conditions that we propose are strictly weaker than those necessary to preserve the branching structure of a transition system used in other approaches to POR for branching time logics [50, 119], increasing the effectiveness of POR.

The experiments with our implementation for solving PBESs are quite promising, although the symbolic reasoning required for static analysis can be a limiting factor (see also Section 4.6). Our results show that, in particular, those instances in which PBESs encode model checking problems involving large state spaces benefit from the use of partial-order reduction. In such cases, a significant size reduction is possible, even when checking complex μ-calculus formulae, and the time penalty of conducting the static analysis is more than made up for by the speed-up in the state space exploration phase.

Outline We start by exploring some related work in Section 6.1. In Section 6.2 we introduce partial-order reduction for parity games, and we proposes a further improvement in Section 6.2.3. Section 6.3 describes how to effectively implement parity-game based POR for PBESs. We present the results of our experiments of using parity-game based POR for PBESs in Section 6.4. We conclude in Section 6.5.

6.1 Related Work

There are several proposals for using partial-order reduction for branching-time logics. Groote and Sellink [57] define several forms of *confluence reduction* and prove which behavioural equivalences (and by extension, which fragments of logics) are preserved. In confluence reduction, one tries to identify internal transitions that can safely be prioritised, leading to a smaller state space. Ramakrishna and Smolka [119] propose a notion that coincides with strong confluence from [57], preserving weak bisimilarity and the corresponding logic, the weak modal μ-calculus.

Similar ideas are presented by Gerth *et al.* in [50]. Their approach is based on the *ample set* method [111] and preserves a relation they call visible bisimulation and the associated logic CTL_{-X}. To preserve the branching structure, they introduce a *singleton proviso* which, contrary to our theory, can greatly impair the amount of reduction that can be achieved (see our Example 6.4, page 98).

Peled [112] applies POR on the product of a transition system and a Büchi automaton, which represents an LTL_{-X} property. The resulting product automaton thus encodes both the transition system and the property, in a way similar to parity games, resulting in a POR approach that is similar to ours. It is important to

note, though, that this original theory is not sound, as discussed in [122]. Kan *et al.* [71] improve on Peled's ideas and manage to preserve all of LTL. To achieve this, they analyse the Büchi automaton that corresponds to the LTL formula to identify which part is stutter insensitive. With this information, they can reduce the state space in the appropriate places and preserve the validity of the LTL formula under consideration.

The work of Bønneland *et al.* [20, 21] is close to ours in spirit: they apply POR to reachability games. Such games can be used for synthesis and for model checking reachability properties. However, the theory presented in [20] is not sound, see Section 5.6.

6.2 Labelled Parity Games

To (partially) resolve the state explosion problem for parity games that encode model checking problems, we apply the partial-order reduction theory from Chapter 5. As we have seen, POR relies on edge labels, here referred to as *events* and typically denoted with the letter j, to categorise the set of edges in a graph and determine independence of edges. In a typical application of POR, such events and edge labellings are deduced from a high-level syntactic description of the graph structure (see also Section 6.3). For now, we tacitly assume the existence of a set of events and edge labellings for parity games and refer to the resulting structures as *labelled parity games*.

Definition 6.1. A *labelled parity game* is a triple $L = (G, \mathcal{S}, \ell)$, where $G = (V, E, \Omega, \mathcal{P})$ is a total parity game, $\mathcal{S}$ is a non-empty set of events and $\ell : \mathcal{S} \to 2^E$ is an edge labelling.

Although we assume here that labelled parity games have a total transition relation, the theory we present can also be applied to parity games that are not total. After all, the stubborn set conditions we use preserve deadlocks. To enable the application of the theory of Chapter 5, we show how a labelled parity game can be interpreted as an LSTS (Definition 5.1).

Definition 6.2. Let $L = (G, \mathcal{S}, \ell)$, with $G = (V, E, \Omega, \mathcal{P})$, be a labelled parity game and $\hat{s} \in V$ some node. Then the corresponding LSTS is $TS = (V, \to, \hat{s}, L)$, where:

- $\to = \{(s, a, t) \mid a \in \mathcal{S}, (s, t) \in \ell(a)\}$; and
- $L(s) = \{(\Omega(s), \mathcal{P}(s))\}$ for all $s \in V$.

As a result, we automatically obtain the notions of invisible events, stutter equivalence for paths in a parity game, a reduction function for labelled parity games, a reduced labelled parity game, *key events* (the counterpart of key actions) and (weak) stubborn sets for labelled parity games. However, the improved conditions **D1'**, **D2w**, **V**, **I** and **L** are not sufficient for preserving the winning player in a labelled parity game. This was noted by Antti Valmari. To avoid situations where, in the

reduced game, one player is forced to move the token to a node belonging to the other player, we need to introduce another condition:

P If there is an event $j \in r(s)$ and a node t such that $s \xrightarrow{j} t$ and $\mathcal{P}(s) \neq \mathcal{P}(t)$, then $r(s) = \mathcal{S}$.

We remark that the above condition was not present in the conference paper that corresponds to this chapter [105] and its correctness theorem (Theorem 1) does not hold.

The remainder of this chapter focusses on weak stubborn sets for labelled parity games, based on the conditions **D1'**, **D2w**, **V**, **I**, **L** and **P**. We show that these are sufficient and, to some extent, necessary conditions for preserving the winning player between the full game and the reduced game.

6.2.1 Necessity of Conditions

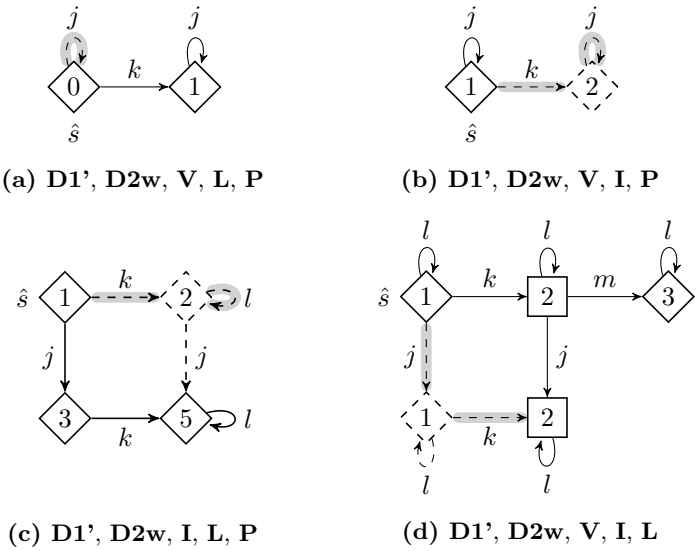
First of all, note that totality of the transition relation is preserved by condition **D2w**. We use the example below to further illustrate the purpose of—and need for—conditions **V**, **I** and **L**. In particular, the example illustrates that the winning player in the original game and the reduced game might be different if one of these conditions is not satisfied.

Example 6.3. First, see the parity games of Figure 6.1. These four games show a reduced game under a reduction function satisfying **D1'** and **D2w** but not **I**, **L**, **V** or **P**, respectively. In each case, we start exploration from the node called $\hat{s}$, using the reduction function to follow the solid edges; consequently, the winning strategy for player $\diamond$ in the original game (highlighted in grey in the figures) is lost. The example in Figure 6.1d, showing the necessity of condition **P**, is due to Antti Valmari.

Now consider the parity game of Figure 6.2, which is inspired by an example from Valmari and Hansen [132]. Here, the condition **L** is replaced by a weaker condition which states that for all visible events j, every *strongly connected component* (SCC) in the reduced game must contain a node s such that $j \in r(s)$. This condition is often called **S** in the literature. Remark that the two leftmost nodes form an SCC and we have $j_1 \in r(\hat{s})$, thus satisfying **S**. However, the cycle consisting of only the bottom-left node does not satisfy condition **L**. □

Note that the games in Figures 6.1a, 6.1b, 6.1c and 6.2 are from a subclass of parity games called *weak solitaire*, illustrating the need for the identified conditions even in restricted settings. A game is *solitaire* iff at most one player can make non-trivial choices. A game is *weak* iff the priorities along all its paths are non-decreasing, *i.e.*, if $s \to t$ then $\Omega(s) \leq \Omega(t)$. The game in Figure 6.1d is weak, but not solitaire. Weak solitaire games can encode the model checking of safety properties, solitaire games can capture logics such as LTL and $\forall$CTL* [60] (the universal fragment of CTL*) and weak games can be used to check CTL properties.

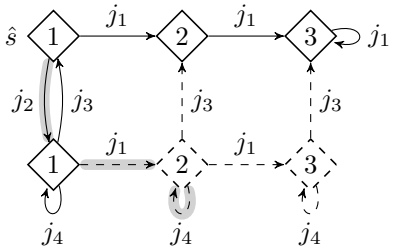
To show that we must use the strengthened condition **D1'** and cannot rely on the original condition **D1**, we revisit the inconsistent labelling problem of Chapter 5 and



(a) D1', D2w, V, L, P **(b) D1', D2w, V, I, P**

(c) D1', D2w, I, L, P **(d) D1', D2w, V, I, L**

Figure 6.1: Three games showing that none of the conditions **V**, **I**, **L** or **P** can be dropped.



D1', D2w, V, I, S, P

Figure 6.2: A parity game that shows condition **L** cannot be weakened to reason about strongly connected components instead of cycles.

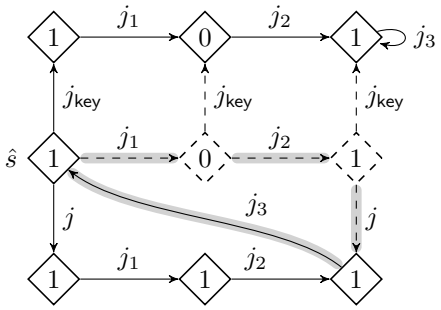


Figure 6.3: A parity game that illustrates the inconsistent labelling problem also occurs in parity games. The winning strategy for player $\diamond$ (highlighted in grey) is lost in the reduced game.

illustrate that it also occurs in the setting of parity games. Consider the parity game in Figure 6.3. Player $\diamond$ wins this game, by moving the token along the edges labelled with $j_1 j_2 j j_3$ (highlighted in grey). In the node $\hat{s}$, the conditions **D1**, **D2w**, **V**, **I**, **L** and **P** allow $r(\hat{s}) = \{j, j_{\mathsf{key}}\}$. Then, the winning strategy for player $\diamond$ is lost in the reduced game.

Before we argue for the correctness of our POR approach in the next section, we shortly demonstrate how our approach improves over existing methods for branching time logics. The conditions **C1-C3** of Gerth *et al.* [50] preserve LTL_{-X} and are similar in spirit to our conditions. However, to preserve the branching structure, needed for preservation of CTL_{-X}, the following *singleton proviso* is introduced:

C4 Either $\mathit{enabled}_G(s) \subseteq r(s)$ or $|r(s)| = 1$.

This extra condition can severely impact the amount of reduction achieved; see the following example.

Example 6.4. Consider the two transition systems below, where $n \geq 1$ is some large natural number.

The cross product of these transition systems contains $(n+1)^2$ states. In the initial state $\hat{s}$, neither $r(\hat{s}) = \{a_1, a'_1\}$ nor $r(\hat{s}) = \{b_1, b'_1\}$ is a valid stubborn set, due to **C4**. However, the labelled parity game constructed using these processes and the μ-calculus formula $\nu X.([\top]X \wedge \mu Y.(\langle\top\rangle Y \vee \langle a_n\rangle \mathit{true}))$, has a very similar shape that *can* be reduced by prioritising transitions that correspond to b_i or b'_i for some $1 \leq i \leq n$. Note that this formula cannot be represented in LTL; condition **C4** is therefore essential for the correctness. □

While several optimisations for CTL_{-X} model checking under POR are proposed in [91], unlike our approach, those optimisations only work for certain classes of CTL_{-X} formulas and not in general.

6.2.2 Correctness

Condition **D2w** suffices, as we already argued, to preserve totality of the transition relation of the reduced labelled parity game. Hence, we are left to argue that the reduced game preserves and reflects the winner of the nodes of the original game; this is formally claimed in Theorem 6.8. We do so by constructing a strategy in the reduced game that mimics the winning strategy in the original game. The plays that are consistent with these two strategies are then shown to be *stutter equivalent*, which suffices to preserve the winner [49]. The example below shows that we cannot simply interpret the parity game as an LSTS and apply Theorem 5.12, since stutter-trace equivalence on the LSTS is weaker than winner equivalence on the labelled parity game.

Example 6.5. Consider the two parity games in Figure 6.4. Observe that the node $\hat{s}$ is won by player $\Box$ in the left game, but player $\Diamond$ wins $\hat{s}'$ in the right game.

Now interpret the same games as LSTSs (disregarding the absence of actions). From the node $\hat{s}$ in the left LSTS, we can perform the traces $(0,\Diamond)(0,\Box)(0,\Box)^\omega$ and $(0,\Diamond)(0,\Box)(1,\Box)^\omega$. Node $\hat{s}'$ on the right has exactly the same traces, so the LSTSs are stutter trace equivalent. □

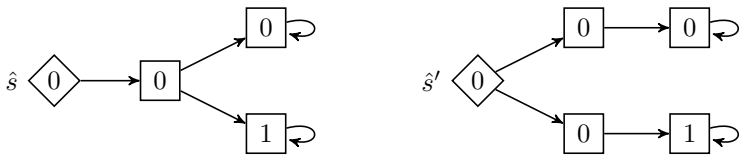


Figure 6.4: Two parity games with a different winner that are stutter-trace equivalent when interpreted as an LSTS.

We introduce a couple of auxiliary lemmata, required for our main correctness theorem. Fix a labelled parity game $L = (G, \mathcal{S}, \ell)$, a node $\hat{s}$, a weak stubborn set r and the reduced labelled parity game $L_r = (G_r, \mathcal{S}, \ell_r)$ induced by r and $\hat{s}$. We assume r and $\hat{s}$ are such that G_r has a finite state space.

Lemma 6.6. *All infinite stutter equivalent paths have the same winner.*

Proof. Since both paths have the same set of priorities that occur infinitely often, they also have the same winner. □

The proof of correctness, *viz.*, Theorem 6.8, uses the alternative paths described by Lemmas 5.10 and 5.11. For convenience, we restate the results here.

Lemma 5.10. *Let r be a weak stubborn set, where condition **D1** is replaced by **D1'**, and $\pi = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$ a path such that $a_1 \notin r(s_0), \ldots, a_n \notin r(s_0)$ and $a \in r(s_0)$. Then, there is a path $\pi' = s_0 \xrightarrow{a}_r s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$ such that $\pi \triangleq \pi'$.*

Lemma 5.11. *Let r be a weak stubborn set, where condition **D1** is replaced by **D1'**, and $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$ a path such that $a_i \notin r(s_0)$ for any a_i that occurs in π. Then, the following holds:*

- *If π is of finite length $n > 0$, there exist an action a_{key}, a state s'_n such that $s_n \xrightarrow{a_{\mathsf{key}}} s'_n$ and a path $\pi' = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'_n$.*
- *If π is infinite, there exists a path $\pi' = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \ldots$ for some action a_{key}.*

In either case, $\pi \triangleq \pi'$.

We need one additional lemma to show that the initial event of a path will be selected for the stubborn set if Lemma 5.10 cannot be applied to that path.

Lemma 6.7. *Let r be a weak stubborn set and $s \xrightarrow{j_1} \xrightarrow{j_2} \cdots \xrightarrow{j_n} s_n$ be a path such that $j_2, \ldots, j_n$ are disabled in s and $j_n \in r(s)$. Then, it must be the case that $j_1 \in r(s)$.*

Proof. By induction. The base case, where $n = 1$, trivially satisfies the condition, since j_n coincides with j_1. For the inductive case, we assume as induction hypothesis that for all paths $s \xrightarrow{j_1} \cdots \xrightarrow{j_l} s_l$ with $l \leq i$, it holds that if $j_2, \ldots, j_l$ are disabled in s and $j_l \in r(s)$, then it holds that $j_1 \in r(s)$. Let $s \xrightarrow{j_1} \cdots \xrightarrow{j_i} s_i \xrightarrow{j_{i+1}} s_{i+1}$ be some path of length $i+1$ that fulfils these same conditions, *i.e.*, $j_2, \ldots, j_i, j_{i+1}$ are disabled in s and $j_{i+1} \in r(s)$. Since the path $s \xrightarrow{j_{i+1} j_1 \cdots j_i}$ is not enabled, condition **D1'** can only be fulfilled by setting $j_l \in r(s)$ for some $l \leq i$. Consequently, we obtain a path $s \xrightarrow{j_1} \cdots \xrightarrow{j_l}$, where $j_2, \ldots, j_l$ are disabled in s and $j_l \in r(s)$. Applying the induction hypothesis to this path yields $j_1 \in r(s)$. □

The following theorem shows that partial-order reduction preserves the winning player in all nodes of the reduced game. Its proof is inspired by [128] and [9, Lemma 8.21], and uses the aforementioned lemmata.

Theorem 6.8. *If L_r has a finite state space then it holds that for every node s in L_r, the winner of s in L_r is equal to the winner of s in L.*

Proof. Let $L = (G, \mathcal{S}, \ell)$, with $G = (V, E, \Omega, \mathcal{P})$, be a labelled parity game and $L_r = (G_r, \mathcal{S}, \ell_r)$, with $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$, a finite reduced game induced by some reduction function r that satisfies conditions **D1'**, **D2w**, **V**, **I**, **L** and **P**. Let player $\bigcirc$ be the winner of some node s in G.

We first consider the case where $\mathcal{P}(s) = \bar{\bigcirc}$. Since none of the outgoing edges of s in E is a winning strategy of $\bar{\bigcirc}$ and $succ_{G_r}(s) \subseteq succ_G(s)$, $\bar{\bigcirc}$ also does not have a winning strategy in s under E_r. Hence, the winner in s is preserved.

Otherwise, if $\mathcal{P}(s) = \bigcirc$, let σ be some winning strategy for $\bigcirc$ in s under E. To prove that $\bigcirc$ also wins node s in G_r, we construct a matching strategy σ' that $\bigcirc$

should follow in G_r. By showing that for every path π in G_r that is consistent with σ', a stutter equivalent path π' can occur in G when following the original strategy σ, we prove that σ' is indeed a winning strategy for $\bigcirc$ starting from s in G_r.

Consider the maximal path $\pi_0 = s \xrightarrow{k_1} s_1 \xrightarrow{k_2} \dots$ that is generated by player $\bigcirc$ moving the token along its own nodes according to σ. The path π_0 is finite if and only if player $\bigcirc$ at some point moves the token to a node owned by player $\bar{\bigcirc}$. In that case, the last node on π_0 is owned by $\bar{\bigcirc}$. We will construct a corresponding path $\hat{\pi}_0$ in the reduced state space. Then, we define σ' such that player $\bigcirc$ moves the token along $\hat{\pi}_0$. Note that although this only defines σ' on a part of the game, the same reasoning can be applied to all other nodes belonging to player $\bigcirc$.

From π_0, we will step-by-step construct new paths π_i that are stutter equivalent to π_0, by shifting events forward (construction of Lemma 5.10) and introducing key events (construction of Lemma 5.11). Since the first step of σ can be trivially imitated if $k_1 \in r(s)$, we henceforth assume that $k_1 \notin r(s)$. Each path π_i is thus of the shape

$$\pi_i = s \xrightarrow{j_1}_r t_1 \xrightarrow{j_2}_r \cdots \xrightarrow{j_i}_r t_i \xrightarrow{k_1} u_0^i \xrightarrow{l_1^i} u_1^i \xrightarrow{l_2^i} \dots$$

where $j_1, \dots, j_i$ are key events in the stubborn set. These can either originate from $k_2, k_3, \dots$, *i.e.*, events that are shifted forward with the construction from Lemma 5.10, or they can be events that are newly introduced using the construction from Lemma 5.11. The events $l_1^i, l_2^i \dots$ represent the remaining subsequence of $k_2, k_3 \dots$, *i.e.*, those events that have not been shifted forward (yet). Note that $j_1, \dots, j_i$ can only contain a visible event if the rest of π_i is invisible (cf. the proofs of Lemma 5.10 and Lemma 5.11).

We distinguish two cases related to whether eventually $k_1 \in r(t_i)$ for some i:

- None of the events $k_1, l_1^i, l_2^i, \dots$ is visible. In that case, player $\bigcirc$ never moves the token to a node belonging to player $\bar{\bigcirc}$ and the path π_0 is infinite. If k_1 is never taken, we can mimic the divergent behaviour of π_i by applying Lemma 5.11 ad infinitum on state $t_i, t_{i+1}, \dots$.

- There is a visible event $m \in \{k_1, l_1^i, l_2^i, \dots\}$. We consider a π_i such that all events that fulfil the requirements of Lemma 5.10 have already been shifted forward, *i.e.*, none of $l_0^i, l_1^i, \dots$ is enabled in t_i. This path exists due to finiteness of E_r. Since the reduced game is finite and the event m is selected at least once on every cycle in the reduced game (condition **L**), there is a $\pi_{i'}$ with $i' \geq i$ such that $m \in r(t_{i'})$. None of the events $l_0^i, l_1^i, \dots$ is enabled in $t_{i'}$ (they did not satisfy Lemma 5.10, after all), therefore – by Lemma 6.7 – it must be the case that $k_1 \in r(t_{i'})$.

In either case, as i goes to ω, we obtain a path $\hat{\pi}_0$. Since condition **P** ensures that reduction only takes places within the nodes belonging to a single player, it must be that all nodes on $\hat{\pi}_0$, except for the last node if $\hat{\pi}_0$ is finite, belong to player $\bigcirc$. This allows us to construct σ' such that it moves the token along $\hat{\pi}_0$. Furthermore, $\hat{\pi}_0$ is stutter equivalent to π_0, since each π_i is stutter equivalent to its predecessor.

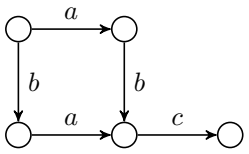
We continue by showing that for every path that is consistent with σ', a corresponding stutter equivalent path that is consistent with σ exists in G. In case $\hat{\pi}_0$ does not contain a node of player $\bar{\bigcirc}$, then $\hat{\pi}_0$ is the only path consistent with σ', by construction of σ', and $\hat{\pi}_0$ is infinite. Its corresponding path in G is π_0.

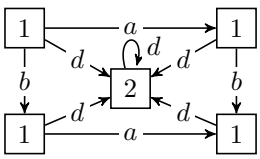
In case $\hat{\pi}_0$ ends in a node owned by $\bar{\bigcirc}$, π_0 is also finite and we reason as follows. From its definition, we know that the last node of π_0, denoted s_n, is owned by player $\bar{\bigcirc}$. There is a path in the original state space from s_n to the last node of $\hat{\pi}_0$, denoted $\hat{s}$, namely along those events of $\hat{\pi}_0$ that were introduced by Lemma 5.11. We call this path π_{key}. We obtain a path $\pi_0\pi_{key}$, which is π_0 extended with invisible key events introduced by Lemma 5.11 in $\hat{\pi}_0$. Since π_{key} contains only invisible events, all nodes on π_{key} are owned by $\bar{\bigcirc}$ and $\pi_0\pi_{key}$ is consistent with σ. Extending a path with finitely many invisible events is permitted under stutter equivalence, hence we conclude that $\hat{\pi}_0$ and $\pi_0\pi_{key}$ are stutter equivalent. □

6.2.3 Optimising D2w

The theory we have introduced so far identifies and exploits rectangular structures in the parity game. This is especially apparent in condition **D1'**. However, parity games obtained from model checking problems also often contain triangular structures, due to the (sometimes implicit) nesting of conjunctions and disjunctions, as the following example demonstrates.

Example 6.9. Consider the process $(a \parallel b) \cdot c$, in which actions a and b are executed in (interleaved) parallel, and action c is executed upon termination of both a and b. The μ-calculus property $\mu X.([\overline{c}]X \wedge \langle\top\rangle \mathit{true})$, also expressible in LTL, expresses that the action c must unavoidably be done within a finite number of steps; clearly this property holds true of the process. Below, the LTS is depicted on the left and a possible parity game encoding of our liveness property on this state space is depicted on the right. The edges in the labelled parity game that originate from the subformula $\langle\top\rangle \mathit{true}$ are labelled with d.





Whereas the state space of the process can be reduced by prioritising a or b, the labelled parity game cannot be reduced due to the presence of a d-labelled edge in every node. For example, if s is the top-left node in the labelled parity game, then $r(s) = \{a, d\}$ violates condition **D1'**, since the path $s \xrightarrow{bd}$ exists, but $s \xrightarrow{db}$ does not. □

In order to deal with games that contain triangular structures, we propose a condition that is weaker than **D2w**.

D2t There is an event $j \in r(s)$ such that for all $j_1 \notin r(s), \ldots, j_n \notin r(s)$, if $s \xrightarrow{j_1} s_1 \xrightarrow{j_2} \cdots \xrightarrow{j_n} s_n$, then either $s_n \xrightarrow{j}$ or there are nodes $s', s'_1, \ldots, s'_n$ such that $s \xrightarrow{j} s' \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \cdots \xrightarrow{j_n} s'_n$ and for all i, $s_i = s'_i$ or $s_i \xrightarrow{j} s'_i$.

Theorem 6.8 holds even for reduction functions satisfying the weak stubborn set conditions in which condition **D2t** is used instead of condition **D2w**. The proof thereof resorts to a modified construction of a mimicking winning strategy that is based on Lemma 6.10, described below, instead of Lemma 5.11.

Lemma 6.10. *Let r be a reduction function satisfying conditions **D1'**, **D2t**, **V**, **I**, **L** and **P**. Suppose $s_0 \xrightarrow{j_1} s_1 \xrightarrow{j_2} \ldots$ such that $j_i \notin r(s_0)$ for every j_i occurring on this path. Then, the following holds:*

- *If the path ends in s_n, there exist a key event j_{key} and nodes $s'_0, \ldots, s'_n$ such that:*
  - *$s_n \xrightarrow{j_{\mathsf{key}}} s'_n$ or $s_n = s'_n$; and*
  - *$s_0 \xrightarrow{j_{\mathsf{key}}}_r s'_0 \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$ and $s_0 \ldots s_n \triangleq s_0 s'_0 \ldots s'_n$.*
- *If the path is infinite, there exists another path $s_0 \xrightarrow{j_{\mathsf{key}}}_r s'_0 \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \ldots$ and $s_0 s_1 \cdots \triangleq s_0 s'_0 s'_1 \ldots$.*

Proof. We follow the same reasoning as in the proof of Lemma 5.11 to conclude the existence of an invisible key event j_{key}.

In case π has finite length n, we derive the existence of $s \xrightarrow{j_{\mathsf{key}}}_r s' \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$ either directly from **D2t** (if $s_n = s'_n$) or from **D1'** (if $s'_n \xrightarrow{j_{\mathsf{key}}}$).

If π is infinite, we distinguish the following cases:

- If $s \xrightarrow{j_{\mathsf{key}} j_1 \cdots j_i} s_i$ for some i, we can trivially extend this path to obtain $\pi' = s \xrightarrow{j_{\mathsf{key}} j_1 \cdots j_i} s_i \xrightarrow{j_{i+1}} \ldots$.
- Otherwise, if there is no i such that $s \xrightarrow{j_{\mathsf{key}} j_1 \cdots j_i} s_i$, we can apply the same reasoning as in the proof of Lemma 5.11.

With the fact that j_{key} is invisible and $s_i \xrightarrow{j_{\mathsf{key}}} s'_i$ or $s_i = s'_i$, we conclude that $\pi \triangleq \pi'$. □

We remark that the concepts of triangular and rectangular structures bear similarities to the concept of weak confluence from [57].

6.3 PBES Solving Using POR

As discussed in Chapter 3, parity games can be used to solve parameterised Boolean equations systems, by first translating them to the SRF normal form. In the remainder of this chapter, we show how to apply POR in the context of solving a PBES. First, we discuss how an SRF-PBES induces a labelled parity game.

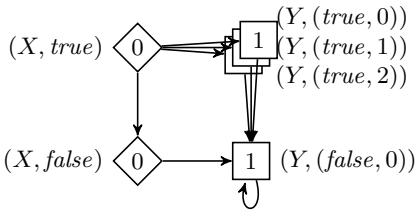


Figure 6.5: Reachable part of the parity game underlying the PBES of Example 6.11, when starting from node (X, true).

Recall that in an SRF-PBES, the right-hand side φ_i in an equation $\sigma_i X_i(d{:}D) = \varphi_i$ is of the shape

$$\bigvee_{j \in J_i} \exists e_j{:}E_j.\, f_j(d, e_j) \wedge X_j(g_j(d, e_j)) \qquad \text{or} \qquad \bigwedge_{j \in J_i} \forall e_j{:}E_j.\, f_j(d, e_j) \Rightarrow X_j(g_j(d, e_j))$$

As a notational convenience, we write J_i to refer to the index set of the formula φ_i, and we assume that the index sets are disjoint for different equations.

Example 6.11. Consider the following PBES in SRF:

$$(\nu X(b{:}B) = (b \wedge X(\mathit{false})) \vee \exists n{:}N.\, n \leq 2 \wedge Y(b, \mathit{if}(b, n, 0)))$$
$$(\mu Y(b{:}B, n{:}N) = \mathit{true} \Rightarrow Y(\mathit{false}, 0))$$

The six nodes in the parity game which are reachable from (X, true) are depicted in Figure 6.5. The horizontally drawn edges all stem from the clause $\exists n{:}N.\, n \leq 2 \wedge Y(b, \mathit{if}(b, n, 0))$. Vertical edges stem from the clause $b \wedge X(\mathit{false})$ (on the left) or the clause $\mathit{true} \Rightarrow Y(\mathit{false}, 0)$ (on the right). The selfloop also stems from the clause $\mathit{true} \Rightarrow Y(\mathit{false}, 0)$. Player $\Box$ wins all nodes in this game, and thus $\mathit{true} \notin [\![\mathcal{E}]\!](X)$. $\Box$

As suggested by the above example, each edge is associated to (at least) one clause in $\mathcal{E}$. Consequently, we can use the index sets J_i to event-label the edges emanating from nodes associated with the equation for X_i. We denote the set of all indices of clauses in $\mathcal{E}$ by $\mathsf{clauses}(\mathcal{E})$, defined as $\mathsf{clauses}(\mathcal{E}) = \bigcup_{X_i \in \mathsf{bnd}(\mathcal{E})} J_i$.

Definition 6.12. Let $\mathcal{E}$ be an SRF-PBES. Then, the labelled parity game corresponding to $\mathcal{E}$ is the structure $(G_{\mathcal{E}}, \mathsf{clauses}(\mathcal{E}), \ell)$, where $G_{\mathcal{E}}$ is the parity game corresponding to $\mathcal{E}$, and, for all $X_i \in \mathsf{bnd}(\mathcal{E})$ and $j \in J_i$, $\ell(j)$ is defined as the set $\{((X_i, v), (X_j, w)) \in E \mid \exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j] \wedge w = [\![g_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j]\}$.

It follows from the definition that the event $j \in J_i$ is *invisible* if $\mathsf{rank}_{\mathcal{E}}(X_i) = \mathsf{rank}_{\mathcal{E}}(X_j)$ and $\mathsf{op}_{\mathcal{E}}(X_i) = \mathsf{op}_{\mathcal{E}}(X_j)$, and *visible* otherwise. Since solving the parity game G corresponding to a PBES $\mathcal{E}$ also yields a solution for $\mathcal{E}$ (Theorem 3.10) and POR preserves the solution of a parity game (Theorem 6.8), it suffices to solve a reduced game G_r to obtain a (partial) solution to $\mathcal{E}$. In model checking, we are

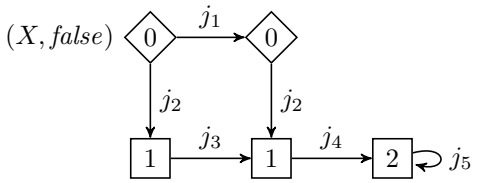


Figure 6.6: Labelled parity game corresponding to the PBES of Example 6.13.

typically only interested in the solution of a single node $X(e)$. Our approach is thus, given a PBES $\mathcal{E}$, to compute a reduced game G_r, starting from $X(e)$, and then determine the winner of $X(e)$ in G_r. If G_r is sufficiently smaller than the full game G and the overhead of computing the reduction function r is limited, this is faster than solving the PBES through the full game G.

6.3.1 Choice of Edge Labelling

The definition of the labelled parity game associated to an SRF-PBES $\mathcal{E}$ (Definition 6.12) associates exactly one event with every clause in $\mathcal{E}$. However, our POR correctness theorem (Theorem 6.8) in principle allows any edge labelling. The choice for the labelling function does influence the amount of reduction that can be achieved. In the extreme cases that every edge has a unique label or all edges are labelled the same, no reduction can be achieved. The below example shows a case for which our current labelling function hinders reduction.

Example 6.13. Consider the SRF-PBES below, where the name of each clause is indicated on the right, and its labelled parity game in Figure 6.6.

$$\begin{aligned} \nu X(b{:}B) &= b \wedge X(\mathit{false}) && (j_1)\\ &\quad \vee Y(b) && (j_2)\\ \mu Y(b{:}B) &= b \Rightarrow Y(\mathit{false}) && (j_3)\\ &\quad \wedge \neg b \Rightarrow X_{\mathit{true}} && (j_4)\\ \nu X_{\mathit{true}} &= X_{\mathit{true}} && (j_5) \end{aligned}$$

Remark that j_2 is visible; the other events are invisible. It is impossible to reduce the parity game: the only viable location is (X, false), but it does not allow stubborn sets smaller than $\{j_1, j_2\}$. Due to condition **I**, we must at least have $r((X, \mathit{false})) = \{j_1\}$. In that case, j_1 is not a key event, since it becomes disabled after j_2; the only way to satisfy **D2w** is to set $r((X, \mathit{false})) = \{j_1, j_2\}$. However, if we label both the j_1 edge and the j_3 edge with a, then it becomes possible to set $r((X, \mathit{false})) = \{a\}$. □

The example suggests that it can be beneficial to use the same event for multiple clauses. Specifying which clauses should correspond to the same event can be done

with an equivalence relation on clauses. The next definition formalises the parity game that follows from such an equivalence relation.

Definition 6.14. Let $\mathcal{E}$ be an SRF-PBES, $\mathcal{R} \subseteq \mathsf{clauses}(\mathcal{E}) \times \mathsf{clauses}(\mathcal{E})$ an equivalence relation on events and $^{\mathsf{clauses}(\mathcal{E})}/_{\mathcal{R}}$ the corresponding set of equivalence classes. Then, the labelled parity game corresponding to $\mathcal{E}$ and $\mathcal{R}$ is the structure $(G_{\mathcal{E}}, {}^{\mathsf{clauses}(\mathcal{E})}/_{\mathcal{R}}, \ell)$, where $G_{\mathcal{E}}$ is the parity game corresponding to $\mathcal{E}$, and, for all $X_i \in \mathsf{bnd}(\mathcal{E})$ and $J \in {}^{\mathsf{clauses}(\mathcal{E})}/_{\mathcal{R}}$, the labelling $\ell(J)$ is defined as follows:

$$\ell(J) = \bigcup_{X_i \in \mathsf{bnd}(\mathcal{E}), j \in J_i \cap J} \{((X_i, v), (X_j, w)) \mid \exists v_j \in \mathbb{E}_j.\, [\![f_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j] \\ \wedge w = [\![g_j(d, e_j)]\!]\delta_0[v/d, v_j/e_j]\}$$

Here, an event $J \in {}^{\mathsf{clauses}(\mathcal{E})}/_{\mathcal{R}}$ is invisible if and only if all its constituting clauses $j \in J$ are invisible. We adopt the strategy to identify two clauses j and j', *i.e.*, set $j \mathcal{R} j'$, iff $f_j = f_{j'}$ and $g_j = g_{j'}$.

6.3.2 Approximating the Conditions

One of the main challenge in implementing POR is that **D1'**, **D2w**/**D2t** and **L** are conditions on the (reduced) state space as a whole and, hence, hard to check locally. To tackle this, we take ideas from literature and show how they apply to PBESs. This allows us to approximate the conditions in such a way that we can construct a stubborn set *on-the-fly*.

From hereon, let $\mathcal{E}$ be a PBES in SRF and $(G, \mathcal{S}, \ell)$, with $G = (V, E, \Omega, \mathcal{P})$, its labelled parity game. The most common local condition for **L** is the *stack proviso* $\mathbf{L}^S$ [111]. This proviso assumes that the state space is explored with *depth-first search* (DFS), and it uses the *Stack* that stores unexplored nodes to determine whether a cycle is being closed. If so, the node will be *fully expanded*, *i.e.*, $r(s) = \mathcal{S}$.

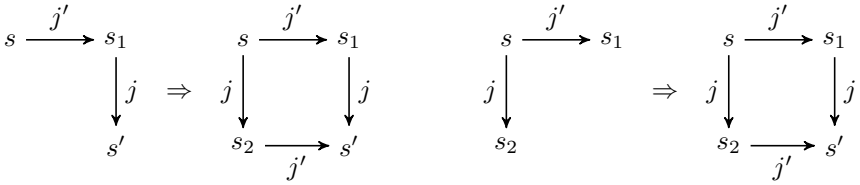
L^S For all nodes $s \in V_r$, either $succ_{G_r}(s) \cap \mathit{Stack} = \emptyset$ or $r(s) = \mathcal{S}$.

We use the *color proviso* [44], which improves on the above condition by keeping track of which nodes on the stack are or will be fully expanded. As a result, less nodes need to be fully expanded, improving reduction potential.

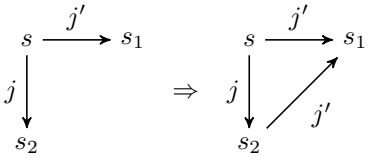
Locally approximating conditions **D1'**, **D2w** and **D2t** requires a static analysis of the PBES. For this, we draw upon ideas from [85] and extend these to properly deal with non-determinism. To reason about which events are independent, we rely on the idea of *accordance*.

Definition 6.15. Let $j, j' \in \mathcal{S}$. We define the *accordance* relations *DNL*, *DNS*, *DNT* and *DNA* on $\mathcal{S}$ as follows:

- j *left-accords* with j' if for all nodes $s, s' \in V$, if $s \xrightarrow{j'j} s'$, then also $s \xrightarrow{jj'} s'$. If j does not left-accord with j', we write $(j, j') \in \mathit{DNL}$.



(a) Left accordance **(b)** Square accordance



(c) Triangle accordance

Figure 6.7: Visual representation of accordance relations.

- j *square-accords* with j' if for all nodes $s, s_1, s_2 \in V$, if $s \xrightarrow{j} s_1$ and $s \xrightarrow{j'} s_2$, then for some $s' \in V$, $s_1 \xrightarrow{j'} s'$ and $s_2 \xrightarrow{j} s'$. If j does not square-accord with j' we write $(j, j') \in DNS$.
- j *triangle-accords* with j' if for all nodes $s, s_1, s_2 \in V$, if $s \xrightarrow{j'} s_1$ and $s \xrightarrow{j} s_2$, then $s_2 \xrightarrow{j'} s_1$. If j does not triangle-accord with j' we write $(j, j') \in DNT$.
- j *accords* with j' if j square-accords or triangle-accords with j'. If j does not accord with j' we write $(j, j') \in DNA$.

Note that DNL and DNT are not necessarily symmetric. An illustration of the left-according, square-according and triangle-according conditions is given in Figure 6.7.

Accordance relations safely approximate the independence of events. The dependence of events, required for satisfying **D2w** can be approximated using Godefroid's *necessary enabling sets* [52].

Definition 6.16. Let j be an event that is disabled in some node s. A *necessary-enabling set* (NES) for j in s is any set $NES_s(j) \subseteq \mathcal{S}$ such that for every path $s \xrightarrow{j_1 \ldots j_n j}$ there is at least one j_i such that $j_i \in NES_s(j)$.

For every node and event there might be more than one NES. In particular, every superset of a NES is also a NES. A larger-than-needed NES may, however, have a negative impact on the reduction that can be achieved. In a PBES with multiple parameters per predicate variable, computing a NES can be done by determining which parameters influence the validity of guards f_j and which parameters are changed in the update functions g_j. A more accurate NES may be computed using techniques to extract a control flow from a PBES [74].

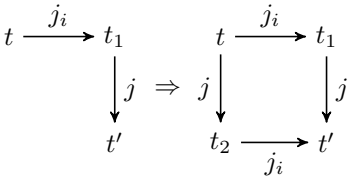
The following three lemmata show how the accordance relations and NES can be used to implement conditions **D1'**, **D2w** and **D2t**, respectively. A combination of Lemma 6.17 and 6.18 in a deterministic setting appeared as Lemma 1 in [85]. Note that as a notational convention we write $R(j)$ to denote the projection $\{j' \mid (j, j') \in R\}$ of a binary relation.

Lemma 6.17. *A reduction function r satisfies **D1'** in node $s \in V$ if for all $j \in r(s)$:*

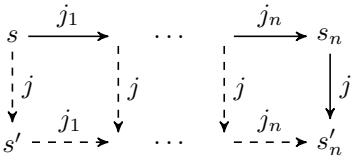
- *if j is disabled in s, then $NES_s(j) \subseteq r(s)$ for some NES_s; and*
- *if j is enabled in s, then $DNL(j) \subseteq r(s)$.*

Proof. Let s be an arbitrary node and let r be a reduction function that satisfies the conditions above. Furthermore, let $s \xrightarrow{j_1 \cdots j_n j} s'_n$ be a path such that $j_1 \notin r(s), \ldots, j_n \notin r(s)$ and $j \in r(s)$. We distinguish the following cases:

- If j is disabled in s, it must be the case that $NES_s(j) \subseteq r(s)$ for some NES_s. However, according to the definition of a necessary-enabling set, at least one of $j_1, \ldots, j_n$ is contained in $NES_s(j)$ and thus in $r(s)$. Since this contradicts our assumption that $j_1 \notin r(s), \ldots, j_n \notin r(s)$, we conclude that the path $s \xrightarrow{j_1 \cdots j_n j} s'_n$ does not exist, and so **D1'** is satisfied.
- If j is enabled in s, it must be that $DNL(j) \subseteq r(s)$. Since that implies $j_1, \ldots, j_n \notin DNL(j)$, it follows that for every j_i and all nodes t, t_1 and t', the following holds:



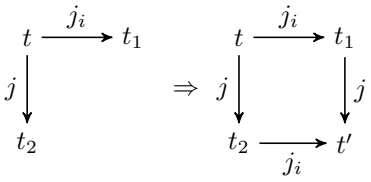
  By inductively applying this implication from right to left on the path $s \xrightarrow{j_1 \cdots j_n} s_n \xrightarrow{j} s'_n$, we derive the existence of the dashed transitions in the figure below.



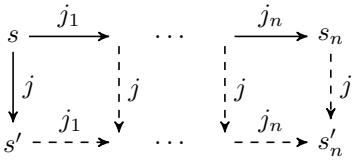
  We conclude that the conditions of **D1'** are satisfied. □

Lemma 6.18. *A reduction function r satisfies **D2w** in a node $s \in V$ if there is an enabled event $j \in r(s)$ such that $DNS(j) \subseteq r(s)$.*

Proof. Let $s \xrightarrow{j_1 \cdots j_n} s_n$ be a path such that $j_1, \ldots, j_n \notin r(s)$ and let $j \in r(s) \cap enabled(s)$ be an event such that $DNS(j) \subseteq r(s)$. We deduce that $j_1, \ldots, j_n \notin DNS(j)$, and thus the following implication holds for all j_i and nodes t, t_1 and t_2:



Applying this inductively from left to right on the transition $s \xrightarrow{j} s'$ and the path $s \xrightarrow{j_1 \cdots j_n} s_n$, we derive the existence of the dashed transitions in the following figure.



Hence, j satisfies the conditions of **D2w**. □

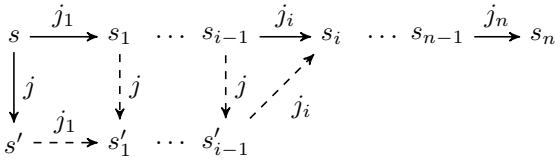
Lemma 6.19. *A reduction function r satisfies **D2t** in a node s if there is an enabled event $j \in r(s)$ such that $DNA(j) \subseteq r(s)$.*

Proof. Let $s \xrightarrow{j_1 \cdots j_n} s_n$ be a path such that $j_1, \ldots, j_n \notin r(s)$ and let $j \in r(s) \cap \mathit{enabled}(s)$ be an event such that $DNA(j) \subseteq r(s)$. We distinguish two cases:

- It holds that $j_1, \ldots, j_n \in DNT(j)$. Since $j_1 \notin r(s), \ldots, j_n \notin r(s)$ and $r(s) \supseteq DNA(j) = DNS(j) \cap DNT(j)$, we can deduce that $j_1, \ldots, j_n \notin DNS(j)$. By following the same reasoning as in the proof of Lemma 6.18, we derive the validity of **D2t**.

- There is an $0 < i \leq n$ such that $j_i \notin DNT(j)$. We consider the smallest such i, *i.e.*, $j_1, \ldots, j_{i-1} \in DNT(j)$. With $j_1 \notin r(s), \ldots, j_n \notin r(s)$ and $r(s) \supseteq DNA(j) = DNS(j) \cap DNT(j)$, we deduce that $j_1, \ldots, j_{i-1} \notin DNS(j)$ and $j_i \notin DNT(j)$.

  By first applying the square-according relation from left to right on the events j and $j_1, \ldots, j_{i-1}$ and then applying the triangle-according relation on j and j_i, we derive the existence of the dashed transitions in the following figure.



  Thus j satisfies the conditions of **D2t**. □

More reduction can be achieved if a PBES is partly or completely 'deterministic', in which case some of the conditions can be relaxed. We say that an event j is *deterministic*, denoted by $det(j)$, if for all nodes $t, t', t'' \in V$, if $t \xrightarrow{j} t'$ and $t \xrightarrow{j} t''$, then also $t' = t''$. This means event-determinism can be characterised as follows:

$$det(j) \text{ iff } [\![f_j]\!]\delta \text{ and } [\![f_j]\!]\delta' \text{ implies } [\![g_j]\!]\delta = [\![g_j]\!]\delta' \text{ for all } \delta, \delta' \text{ with } \delta(d) = \delta'(d).$$

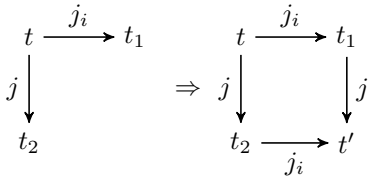
The following lemma specialises Lemma 6.17 and shows how knowledge of deterministic events can be applied to potentially improve the reduction.

Lemma 6.20. *A reduction function r satisfies **D1'** in a node s if for all $j \in r(s)$:*

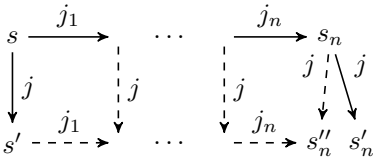
- *if j is disabled in s, then $NES_s(j) \subseteq r(s)$ for some NES_s; and*
- *if $det(j)$ and j is enabled in s, then $DNS(j) \subseteq r(s)$ or $DNL(j) \subseteq r(s)$.*
- *if $\neg det(j)$ and j is enabled in s, then $DNL(j) \subseteq r(s)$.*

Proof. Let s be an arbitrary node and let r be a reduction function that satisfies the conditions above. For the cases where j is disabled or j is enabled and $DNL(j) \subseteq r(s)$, see the proof of Lemma 6.17. Here, we only consider the new case where j is deterministic and enabled in s and $DNS(j) \subseteq r(s)$.

Let $s \xrightarrow{j_1 \cdots j_n} s_n \xrightarrow{j} s'_n$ be a path such that $j_1 \notin r(s), \ldots, j_n \notin r(s)$ and $j \in r(s)$ and let $s \xrightarrow{j} s'$. The following implication holds for all j_i and nodes t, t_1 and t_2:



Applying this inductively from left to right on the transition $s \xrightarrow{j} s'$ and the path $s \xrightarrow{j_1 \cdots j_n} s_n$, we deduce the existence of the dashed transitions for some node s''_n.



Since j is deterministic it follows that $s'_n = s''_n$, and thus **D1'** is satisfied. □

Since relations *DNS* and *DNL* are incomparable we cannot decide *a priori* which should be used for deterministic events. However, Lemma 6.20 permits choosing one of the accordance sets on-the-fly. This choice can be made based on a heuristic function, similar to the function for NESs proposed in [85].

Both the accordance relations and the NESs can be computed based on the condition f_j and update expression g_j associated with each clause. This computation can also take into account in which equation the clause occurs and to which predicate variable it leads for more accurate results. Since the clause identification strategy we adopted relates clauses with the same conditions and update expressions, the analysis of clauses trivially carries over to events. The reduced number of events has as additional benefit that less static analysis is required to construct the accordance and NES relations.

6.4 Experiments

We implemented the ideas from the previous section in a prototype tool, called `pbespor`, as part of the mCRL2 toolset [26]; it is written in C++. Our tool converts a given input PBES to a PBES in SRF, runs a static analysis to compute the accordance relations (see Section 6.3), and uses a depth-first exploration to compute the parity game underlying the PBES in SRF. The static analysis relies on an external SMT solver (we use Z3 [38] in our experiments). Experiments are conducted on a machine with an Intel Xeon 6136 CPU @ 3 GHz, running mCRL2 with Git commit hash `dd36f98875`.

To measure the effectiveness of our implementation, we analysed seven mCRL2 models[1]: Anderson's mutual exclusion protocol [5], the dining philosophers problem, the gas station problem [63], Hesselink's handshake register [65], Le Lann's leader election protocol [88], Milner's Scheduler [98] and the Krebs cycle of ATP production in biological cells (model inspired by [110]). Most of these models are scalable. Each model is subjected to one or more requirements phrased as a μ-calculus formulae. Where possible, Table 6.1 provides a CTL* formula that captures the essence of the requirement.

We analyse the effectiveness of our partial-order reduction technique by measuring the reduction of the size of the state space, and the time that is required to generate the state space. Since the static analysis that is conducted can require a non-negligible amount of time, we pay close attention to the various forms of static analysis that can be conducted. In particular, we compare the total time and effectiveness (in terms of reduction) of running the following static analysis:

- computing left-accordance (*DNL*) vs. over-approximating it with all events.
- computing a NES vs. over-approximating it with the set of all events $\mathcal{S}$.
- using **D2w** vs. the use of **D2t** (*i.e.*, use Lemma 6.18 vs. Lemma 6.19);

As a baseline for comparisons, we take a basic static analysis (over-approximated DNL, over-approximated NES, **D2w**), see column 'basic' in Table 6.1. In order to guarantee termination of the static analysis phase, we set a timeout of 200ms per

[1] The models are archived online at `https://doi.org/10.5281/zenodo.3602969`.

Table 6.1: Runtime (analysis + exploration; in seconds) and number of states when exploring either the full state space or the reduced state space, for four different static analysis approaches. Figures printed in boldface indicate which of the additional static analyses is able to achieve the largest reduction over 'basic' (if any).

		full		basic		+DNL		+NES		+D2t	
model	property	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time
gas station.c3	$\exists\Diamond$*accept*	1 197	0.14	1 077	0.98	1 077	2.48	1 077	1.87	**735**	1.62
gas station.c3	$\exists\Box\exists\Diamond$*pumping*	1 261	0.15	967	0.98	967	2.61	967	1.99	967	1.72
gas station.c3	no deadlock	1 197	0.18	735	0.95	735	2.52	735	2.04	735	1.52
scheduler8	no deadlock	3 073	0.29	34	0.19	34	0.70	34	0.51	34	0.35
scheduler10	no deadlock	15 361	1.65	42	0.25	42	0.90	42	0.65	42	0.42
anderson.5	$\forall\Diamond$*cs*	23 597	4.59	2 957	2.85	2 957	6.47	2 957	3.89	2 957	4.61
hesselink	cache consistent	91 009	5.28	82 602	8.19	83 602	12.12	81 988	9.00	**71 911**	8.51
dining10	no deadlock	154 451	17.90	4 743	0.76	4 743	1.61	4 743	1.42	4 743	1.02
krebs.3	$\forall\Diamond$*energy*	238 877	24.38	232 273	24.59	232 273	25.62	**209 345**	21.73	232 273	24.42
gas station.c6	$\exists\Diamond$*accept*	186 381	38.00	150 741	40.55	150 741	45.50	150 741	43.16	**75 411**	21.40
gas station.c6	$\exists\Box\exists\Diamond$*pumping*	192 700	38.63	114 130	27.35	114 130	31.42	114 130	30.49	114 130	29.74
gas station.c6	no deadlock	186 381	42.50	75 411	21.03	75 411	24.88	75 411	24.01	75 411	23.02
scheduler14	no deadlock	344 065	53.14	58	0.37	58	1.31	58	0.97	58	0.61
hesselink	$\forall\Box(wr\Rightarrow\exists\Diamond fin)$	1 047 233	61.02	1 013 441	82.44	1 013 441	86.49	1 013 441	84.59	**791 273**	61.56
hesselink	$\forall\Box(wr\Rightarrow\forall\Diamond fin)$	1 047 232	70.14	791 320	64.05	791 374	66.53	**749 936**	62.98	791 268	67.59
krebs.4	$\forall\Diamond$*energy*	1 047 406	124.30	971 128	117.38	971 128	117.41	**843 349**	101.51	971 128	117.41
lann.5	consistent data	818 104	142.38	818 104	170.18	818 104	175.87	818 104	177.78	**761 239**	155.22
anderson.5	no deadlock	689 901	142.63	257 944	73.62	257 672	79.91	257 711	78.67	257 918	76.47
lann.5	no data loss	1 286 452	199.74	453 130	73.28	453 130	77.31	453 130	74.40	453 130	75.52
dining10	$\forall\Box\forall\Diamond$*eat*	1 698 951	225.10	101 185	12.37	101 056	13.55	101 238	13.01	101 022	12.69
anderson.7	$\forall\Diamond$*cs*	3 964 599	1 331.91	124 707	63.83	124 707	73.87	124 707	68.67	124 707	69.68
scheduler18	no deadlock	7 077 889	1 574.25	74	0.54	74	1.80	74	1.40	74	0.85

formula that is sent to the solver. Table 6.1 reports on the statistics we obtained for exploring the full state space and the four possible POR configurations described above; the table is sorted with respect to the time needed for a full exploration. The time we list consists of the time needed to conduct the analysis plus the time needed for the exploration.

For most small instances, the time required for static analysis dominates any speed-up gained by the state space reduction. When the state spaces are larger, achieving a speed-up becomes more likely, while the highest overhead suffered by 'basic' is 55% (Hesselink, cache consistency). Significant reduction can be achieved even for non-trivial properties, such as 'lann.5' with 'no data loss'. Scheduler is an extreme case: its processes have very few dependencies, leading to an exponential reduction, both in terms of the state space size and in terms of time. In several cases, the use of a NES or **D2t** brings extra reduction (highlighted in bold). Moreover, the extra time required to conduct the additional analysis seems limited. The use of DNL, on the other hand, never pays off in our experiments; it even results in a slightly larger state space in two cases.

We note that there are also models, not listed in Table 6.1, where our static analysis does not yield any useful results and no reduction is achieved. Even if in such cases a reduction would be possible in theory, the current static analysis engines are unable

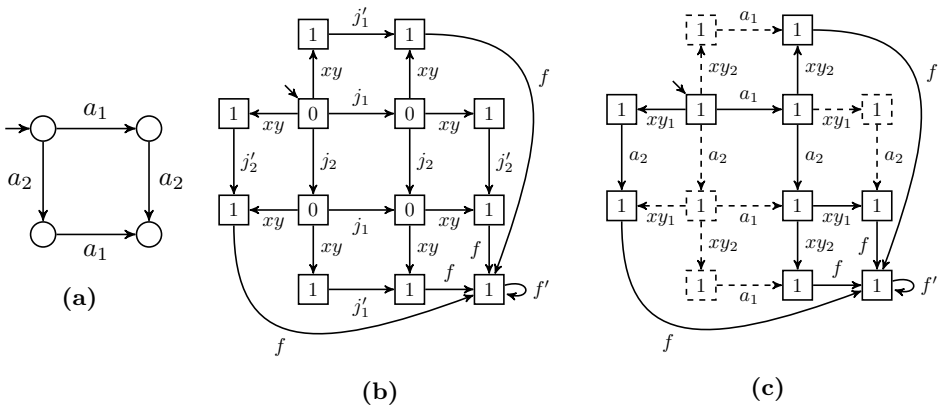


Figure 6.8: An LTS and two parity games showing that, although the LTS can be reduced, it is not always possible to reduce the corresponding parity game. After tweaking the edge labelling, some reduction is possible.

to deal with the more complex data types often used in such models; *e.g.*, recursively defined lists or infinite sets, represented symbolically with higher-order constructions. This calls for further investigations into static analysis theories that can effectively deal with complex data.

Furthermore, we remark that all the models we listed here contain *symmetry*: two or more processes that show almost the same behaviour. This may mean that our results cannot be generalised to non-symmetric models, although [42] suggests that partial-order reduction and symmetry reduction exploit different aspects of a model.

Finally, we point out that in the case of, *e.g.*, the dining philosophers problem, the relative reduction under the 'no deadlock' property is much better than under the '$\forall\Box\forall\Diamond eat$' property. This demonstrates the impact properties can have on the reductions achievable and the importance of the way edges are labelled. We explain this in the following example.

Example 6.21. Consider the PBES below, which encodes the formula $\nu X.([\top]X \wedge \forall i.\, \mu Y.([\overline{a_i}]Y \wedge \langle\top\rangle \mathit{true}))$ on the LTS of Figure 6.8a. For reference, in Table 6.1, we denoted formulae of this shape as $\forall\Box\forall\Diamond a_i$. Below, the names of each of the clauses is indicated on the right.

$$\begin{aligned} \nu X(b_1, b_2{:}B) = {} & b_1 \Rightarrow X(\mathit{false}, b_2) && (j_1)\\ & \wedge\, b_2 \Rightarrow X(b_1, \mathit{false}) && (j_2)\\ & \wedge\, \forall b{:}B.\, Y(b_1, b_2, b) && (xy) \end{aligned}$$

$$\begin{aligned} \mu Y(b_1, b_2, b{:}B) = (b \wedge b_1) &\Rightarrow Y(\mathit{false}, b_2, b) && (j_1') \\ \wedge\ (\neg b \wedge b_2) &\Rightarrow Y(b_1, \mathit{false}, b) && (j_2') \\ \wedge\ \neg(b \wedge b_1 \vee \neg b \wedge b_2) &\Rightarrow X_{\mathit{false}} && (f) \\ \mu X_{\mathit{false}} = X_{\mathit{false}} & && (f') \end{aligned}$$

The event xy represents the transition from fixpoint X into Y, which does not involve an action from the LTS. Note that the complete state space is encoded once in the fixpoint X and twice in Y, albeit with a subset of the transitions. In the corresponding labelled parity game, depicted in Figure 6.8b, no reduction can be achieved. □

To achieve reduction in this example, we need to do three things. First, the quantifier in the clause xy needs to be unfolded, yielding two clauses $Y(b_1, b_2, \mathit{false})$ (name xy_1) and $Y(b_1, b_2, \mathit{true})$ (name xy_2). Furthermore, we should identify clauses j_1 and j_1' (resp. j_2 and j_2'); the resulting event is called a_1 (resp. a_2). Lastly, we have to change the fixpoint of X to ensure xy_1 and xy_2 are invisible. Remark that this does not change the solution of the PBES. In that case, four nodes can be eliminated, see Figure 6.8c.

In the experiments, we achieve the identification of clauses j_1 and j_1' (resp. j_2 and j_2') by partially instantiating the PBES. This procedure is implemented in the mCRL2 tool `pbesinst`. In the example above, instantiating parameter $b{:}B$ of Y results in two new equations:

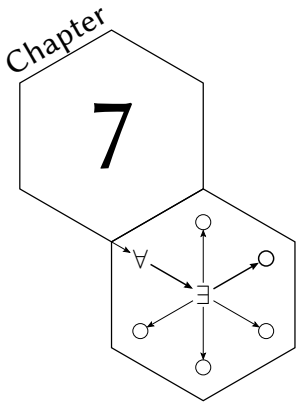
$$\begin{aligned} \mu Y_{\mathit{true}}(b_1, b_2) = b_1 &\Rightarrow Y_{\mathit{true}}(\mathit{false}, b_2) && (j_1) \\ \wedge\ \neg b_1 &\Rightarrow X_{\mathit{false}} \\ \mu Y_{\mathit{false}}(b_1, b_2) = b_2 &\Rightarrow Y_{\mathit{false}}(b_1, \mathit{false}) && (j_2) \\ \wedge\ \neg b_2 &\Rightarrow X_{\mathit{false}} \end{aligned}$$

The clauses in these equations can be identified with the first two clauses in the right-hand side of X with our original strategy based on syntactic equivalence of the condition and update expressions.

6.5 Conclusion

We have presented an approach for applying partial-order reduction on parity games. This has two main advantages over POR applied on LTSs or Kripke structures: our approach supports the full modal μ-calculus, not just a fragment thereof, and the potential for reduction is greater, because we do not require a singleton proviso. Furthermore, we have shown how the ideas can be implemented with PBESs as a high-level representation. In future work, we aim to gain more insight into the effect of identifying events across PBES equations in several ways. We also want to investigate the possibility of solving a reduced parity game while is it being constructed. In certain cases, one may be able to decide the winner of the original game from this partial solution.

Quantifier Manipulation in PBESs



The reduction techniques we have studied so far rely heavily on semantics: they reason about (the existence of) transitions in the parity game of a PBES. Another way to reduce the underlying graph structure is the application of *syntactic* transformation techniques: procedures that modify the equations and formulae in a PBES and do not manipulate the underlying graph directly. The SRF and CRF transformations of Chapter 3 are a nice example of this (although they do not reduce the dependency graph).

The existing syntactic reductions for PBESs can be divided into two categories. Firstly, there are a number of techniques that manipulate predicate formulae, but do not consider the dependencies between equations. Secondly, there are several techniques [74, 106] which perform data flow analysis across all equations. As a result, they obtain knowledge on the interdependence of data parameters and also on the parameter's influence on each right-hand side. This knowledge can be used for syntactic transformations that aim to reduce the underlying graph.

However, a common shortcoming among these techniques is that they only partially deal with quantifiers. We address this in the current chapter, with the goal of improving the reduction. First of all, we propose a technique called *quantifier propagation*, which can move a quantifier $\forall d{:}D$ or $\exists d{:}D$ that surrounds a predicate variable instance $X(e)$ to the corresponding right-hand side φ_X. This prevents the

instantiation of $X(e)$ for every possible value of d, thus reducing the underlying parity game. Quantifier propagation can be seen as an extension of the *quantifier-inside rewriter*, which distributes quantifiers over other logical operators whenever possible. As a side result, we contribute an improvement of the distribution of universal quantification over disjunction (and, dually, of existential quantification over conjunction).

This basic quantifier propagation technique can sometimes reduce the underlying graph structure of a PBES, but is in some cases not able to completely eliminate a quantifier from a PBES. Furthermore, it can only be applied to one subformula at a time and not straightforward to decide where it should be applied. Thus, we propose a static analysis technique to identify which predicate variables always occur in the scope of the same quantifiers. The result of this analysis allows us to apply *global propagation* at once throughout the whole PBES. Our approach generalises the constant elimination algorithm from [106] by taking quantifiers into consideration. Experiments with two PBESs from literature indicate that global propagation can work well in practice and has the potential to reduce an underlying parity game of infinite size to a finite-sized parity game.

If, during the static analysis required for global propagation, we can determine that the occurrence of a predicate variable $X(e)$ is not relevant in a right-hand side φ, the instance $X(e)$ does not need to be considered. The relevance of a predicate variable instance $X(e)$ is captured in the concept of a *guard*. More precisely, a guard is a predicate formula that expresses when the occurrence of a predicate variable $X(e)$ is relevant to the truth value of its containing formula. Guards are thus a valuable tool in our analysis, and likewise in the static analysis performed in [74, 106]. However, [106] only shows how to extract guards for predicate formulae in PFNF (see Chapter 3). The definitions of [74] can compute guards for arbitrary predicate formulae, but only partly take quantifiers into account. Furthermore, the result may depend on the nesting structure of the formula, indicating that the result is not always optimal. For example, the computation yields different guards for $f \wedge (X \wedge Y)$ and $(f \wedge X) \wedge Y$. We show how to compute guards that are stronger than the guards of [74], which enables extracting more information. Furthermore, we provide a detailed proof that our guards can be applied compositionally. We also investigate *exact guards*, and show that they are not compositional. This makes it difficult to compute them efficiently for large predicate formulae.

Overview In Section 7.1, we discuss several related works on analysis of PBESs. Then, Section 7.2 introduces new concepts and notation, which are required in the remainder of the chapter. We motivate our approach with an example in Section 7.3. Then, Section 7.4 shows how to perform quantifier propagation. The ideas are extended to global propagation in Section 7.5. In Section 7.6, we discuss guards and how they can be computed efficiently. Section 7.7 gives details of an implementation of global propagation and shows how it may perform in practice by means of a small experiment. Finally, Section 7.8 presents concluding remarks and suggestions for

future work.

7.1 Related Work

We informally describe several syntactic transformation techniques for PBESs that have been proposed in previous works. The first technique is *constant elimination* [106], which attempts to identify parameters whose value never changes. Constant elimination is performed relative to a closed *target expression* κ that one wants to solve. The analysis starts by taking constant values from κ and propagating them through the PBES to determine whether their value will change. If not, an invariant [59, 107] is derived that can be used to simplify the PBES. Constant elimination by itself never results in state space reduction, but the resulting simplifications can speed up PBES instantiation and also enable other reduction techniques.

The same paper also discusses *parameter elimination* [106], which tries to identify a set of parameters that do not influence any of the conditions in the PBES. This is achieved with a fixpoint procedure on an influence graph over parameters. Initially, only parameters that occur in a Boolean term b in the right-hand side of their equation are marked influential. The influence graph relates two parameters d_1 and d_2 if d_2 occurs in an update of d_1. Parameters that are not (transitively) influential can be safely eliminated from the PBES, without affecting its solution.

Very much related to parameter elimination, is the state graph technique of [74], which is inspired by a similar approach for LPSs [118]. This analysis consists of two phases: it first attempts to distinguish *control flow* parameters—parameters which can take a limited number of values and are updated deterministically—from data parameters. Based on knowledge of the values the control flow parameters can take on, a control flow graph can be reconstructed. Then, it proceeds by analysing which data parameters are influential in each of the locations of the control flow graph. If a data parameter is not influential in a certain control flow location, it may be reset to a fixed value when taking a transition to that location.

7.2 Preliminaries

To understand the theory in this chapter, we first need to introduce several new concepts. First, the analysis and transformation techniques that we present in Sections 7.4 and 7.5 require reasoning about parameters of predicate variables and their occurrences in right-hand sides. Therefore, the theory we present in those sections no longer assumes that predicate variables have only a single parameter. Consequently, we need to reason about finite sequences of parameters and expressions, which we denote with a bold font. Furthermore, we often need to indicate to which equation fixpoints, parameters and right-hand sides are associated, so we assume each equation has the shape $\sigma_X X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X$, unless specified otherwise.

Sequences We adopt the following notation for sequences. Firstly, the empty sequence is denoted ϵ. Given a sequence $\mathbf{v}$, $\mathbf{v}[i]$ denotes its ith element. If $\mathbf{v}$ is a sequence of length n, then a strictly increasing, total function $f : \{1, \ldots, n'\} \to \{1, \ldots, n\}$ characterises a *subsequence* $\mathbf{v}'$ of length $n' \leq n$, *viz.*, $\mathbf{v}'[i] = \mathbf{v}[f(i)]$ for all $1 \leq i \leq n'$. The fact that $\mathbf{v}'$ is a subsequence of $\mathbf{v}$ with characterising function f is denoted $\mathbf{v}' \sqsubseteq_f \mathbf{v}$. We omit the subscript f if it is not relevant. Furthermore, given two sequences $\mathbf{v}'$ and $\mathbf{v}$ such that $\mathbf{v}' \sqsubseteq_f \mathbf{v}$, we write $\mathbf{v}[j] \in \mathbf{v}'$ iff there is an i such that $f(i) = j$. We use list comprehensions to filter a sequence and obtain a subsequence, *e.g.*, $[\mathbf{v}[i] \mid i \mod 2 = 0]$ is the unique subsequence that contains the elements with an even index in $\mathbf{v}$ and only those. Lastly, we can calculate the complement of a subsequence, notation $\mathbf{v} \setminus \mathbf{v}'$, which is defined as $[\mathbf{v}[i] \mid \mathbf{v}[i] \notin \mathbf{v}']$. To concatenate two sequences, we use $+\!\!+$.

Quantifiers We generally use $\mathsf{Q} \in \{\forall, \exists\}$ to denote an arbitrary quantifier and $\mathbf{Q} \in (\{\forall, \exists\} \times \mathcal{V})^*$ to denote a finite sequence of quantified variables. Each element in such a sequence is a pair of a quantifier and a sorted variable, which is not necessarily unique in $\mathbf{Q}$. In semantics, a sequence of quantifiers is denoted $\boldsymbol{\mathcal{Q}}$. We use the functions q and v to extract the quantifier and variable from quantified variables respectively, *i.e.*, $\mathsf{q}((\mathsf{Q}, d)) = \mathsf{Q}$ and $\mathsf{v}((\mathsf{Q}, d)) = d$. These functions are also lifted to sequences such that $\mathsf{q}(\epsilon) = \epsilon$ and $\mathsf{v}(\epsilon) = \epsilon$ and for a non-empty sequence $\mathbf{Q}$, we have $\mathsf{q}(\mathbf{Q})[i] = \mathsf{q}(\mathbf{Q}[i])$ and $\mathsf{v}(\mathbf{v})[i] = \mathsf{v}(\mathbf{v}[i])$ for all $1 \leq i \leq |\mathbf{Q}|$. The same functions can also be applied to semantic quantifier sequences. A sequence of quantified variables $\mathbf{Q}$ can be projected on a set of variables V, which yields the subsequence $\mathbf{Q}_{\downarrow V} = [\, \mathbf{Q}[i] \mid \mathsf{v}(\mathbf{Q}[i]) \in V \,]$. We overload the projection operator with expressions, such that $\mathbf{Q}_{\downarrow e} = \mathbf{Q}_{\downarrow \mathsf{vars}(e)}$.

We often consider subformulae of the shape $\mathbf{Q}.\, X(\mathbf{e})$, with $\mathbf{e}$ a sequence of expressions, which we call *quantified predicate variable instance* (QPVI). A QPVI $X(\mathbf{e})$ without quantifiers is simply called a *predicate variable instance* (PVI). The set of all PVIs that occur in a formula φ is denoted $\mathsf{iocc}(\varphi)$.

Environment updates and substitutions We allow sequences of values and sequences of variables in data environment updates, *i.e.*, given a sequence of distinct variables $\mathbf{d}$ and a sequence of values $\mathbf{v}$ of equal length and equal sort, $\delta[\mathbf{v}/\mathbf{d}]$ is the data environment $\delta[\mathbf{v}[1]/\mathbf{d}[1], \ldots, \mathbf{v}[n]/\mathbf{d}[n]]$. Furthermore, we use the same notation for syntactic predicate formula *substitutions*: $\varphi[\psi'/\psi]$ is the formula φ where every occurrence of ψ as a subformula of φ is replaced by ψ'. To perform several substitutions at once, we allow the substitution of sequences of expressions, *e.g.*, $\varphi[\mathbf{e}'/\mathbf{e}]$, or we write $\varphi[f(\psi)/\psi]_{\psi \in \Psi}$, where Ψ is a set or sequence of expressions. Remark that a predicate formula can be viewed as a function, just like a predicate variable. This motivates the shorthand notation $\varphi[\psi/X]$ for $\varphi[\psi[\mathbf{e}/\mathbf{d}_X]/X(\mathbf{e})]_{X(\mathbf{e}) \in \mathsf{iocc}(\varphi)}$. Similarly, we allow one predicate variable to be replaced by another, $\varphi[Y/X]$ denotes $\varphi[Y(\mathbf{e})/X(\mathbf{e})]_{X(\mathbf{e}) \in \mathsf{iocc}(\varphi)}$. Finally, we extend substitutions to PBESs, such that $\mathcal{E}[\psi'/\psi]$ is the PBES $\mathcal{E}$ where ψ is replaced by ψ' in every right-hand side of $\mathcal{E}$.

Predicate formula quantifier manipulation We describe two relatively simple quantifier manipulation techniques that operate on predicate formulae. These follow from well-known results in logic, see *e.g.* [69]. Some of the examples in this chapter rely on these techniques to simplify the right-hand sides in a PBES. Firstly, there is the *one-point rule*, also known as the substitution rule, which can be used to eliminate quantifiers of which only one value of the quantified variable is relevant for its evaluation, as formalised below:

$$[\![\forall d{:}D.\, d \neq d_1 \vee \varphi]\!]\eta\delta = [\![\varphi[d_1/d]]\!]\eta\delta$$
$$[\![\exists d{:}D.\, d = d_1 \wedge \varphi]\!]\eta\delta = [\![\varphi[d_1/d]]\!]\eta\delta$$

We refer to the rewriter that implements this rule as the *one-point rewriter*.

Secondly, the *quantifier-inside rewriter* [135] distributes quantifiers over other operators whenever possible. It relies on an analysis of free variables to determine when a universal quantifier can be distributed over disjunction and, dually, when an existential quantifier can be distributed over conjunction. The definition is as follows.

$$\begin{array}{ll}
\mathsf{qi}(b) & = b \\
\mathsf{qi}(\neg\varphi) & = \neg\mathsf{qi}(\varphi) \\
\mathsf{qi}(\varphi \wedge \psi) & = \mathsf{qi}(\varphi) \wedge \mathsf{qi}(\psi) \\
\mathsf{qi}(\varphi \vee \psi) & = \mathsf{qi}(\varphi) \vee \mathsf{qi}(\psi) \\
\mathsf{qi}(\forall W.\varphi) & = \mathsf{qi}_\forall(W, \mathsf{qi}(\varphi)) \\
\mathsf{qi}(\exists W.\varphi) & = \mathsf{qi}_\exists(W, \mathsf{qi}(\varphi)) \\
\mathsf{qi}(X(e)) & = X(e)
\end{array}$$

The definition of $\mathsf{qi}_\forall$ is:

$$\begin{array}{ll}
\mathsf{qi}_\forall(V, b) & = \forall V \cap \mathsf{vars}(b).\, b \\
\mathsf{qi}_\forall(V, \neg\varphi) & = \neg\mathsf{qi}_\exists(V, \varphi) \\
\mathsf{qi}_\forall(V, \varphi \wedge \psi) & = \mathsf{qi}_\forall(V, \varphi) \wedge \mathsf{qi}_\forall(V, \psi) \\
\mathsf{qi}_\forall(V, \bigvee_i \varphi_i) & = \begin{cases}
\forall V \cap \mathsf{vars}(\bigvee_i \varphi_i).\, \bigvee_i \varphi_i & \text{if } \psi = \mathit{false} \\
\forall V \cap \mathsf{vars}(\varphi) \cap \mathsf{vars}(\psi).\ (\mathsf{qi}_\forall((V \cap \mathsf{vars}(\varphi)) \setminus \mathsf{vars}(\psi), \varphi) \vee \mathsf{qi}_\forall((V \cap \mathsf{vars}(\psi)) \setminus \mathsf{vars}(\varphi), \psi)) & \text{otherwise}
\end{cases} \\
& \quad \text{where} \\
& \quad W = \mathsf{vars}(\varphi_j) \cap V \text{ for the smallest } j \text{ such that } |W| \text{ is minimal,} \\
& \quad \varphi = \bigvee \{\varphi_i \mid \mathsf{vars}(\varphi_i) \cap V \subseteq W\} \\
& \quad \psi = \bigvee \{\varphi_i \mid \mathsf{vars}(\varphi_i) \cap V \not\subseteq W\} \\
\mathsf{qi}_\forall(V, \forall W.\varphi) & = \mathsf{qi}_\forall(V \cup W, \varphi) \\
\mathsf{qi}_\forall(V, X(e)) & = \forall V \cap \mathsf{vars}(e).\, X(e)
\end{array}$$

The idea behind the fourth case, where the universal quantifier is distributed over disjunction, is as follows. We aim to partition the set of disjuncts $\{\varphi_1, \ldots, \varphi_n\}$ into two parts φ and ψ, such that the set $V \cap \mathsf{vars}(\varphi) \cap \mathsf{vars}(\psi)$ is small. This ensures a high number of variables can be pushed further inside. We achieve this through a linear search over the disjuncts φ_i, and selecting a minimal set $W = \mathsf{vars}(\varphi_j) \cap V$.

The set of variables W partitions the disjunction: each φ_i is placed into φ or ψ based on which variables it shares with W. If all disjuncts contain the same free variables, then ψ equals *false* and no distribution is possible. Otherwise, we bind each quantified variable based on whether it occurs only in φ, only in ψ or in both.

The dual rewrite function $\mathsf{qi}_\exists$ is defined by

$$
\begin{array}{lll}
\mathsf{qi}_\exists(V, b) & = \exists V \cap \mathsf{vars}(b).\, b & \\
\mathsf{qi}_\exists(V, \neg\varphi) & = \neg \mathsf{qi}_\forall(V, \varphi) & \\
\mathsf{qi}_\exists(V, \bigwedge_i \varphi_i) & = \begin{cases} \exists V \cap \mathsf{vars}(\bigwedge_i \varphi_i).\, \bigwedge_i \varphi_i & \text{if } \psi = \mathit{true} \\ \exists V \cap \mathsf{vars}(\varphi) \cap \mathsf{vars}(\psi).\ (\mathsf{qi}_\exists((V \cap \mathsf{vars}(\varphi)) \setminus \mathsf{vars}(\psi), \varphi) \vee \mathsf{qi}_\exists((V \cap \mathsf{vars}(\psi)) \setminus \mathsf{vars}(\varphi), \psi)) & \text{otherwise} \end{cases} \\
 & \quad \text{where} & \\
 & \quad W = \mathsf{vars}(\varphi_j) \cap V \text{ for the smallest } j \text{ such that } |W| \text{ is minimal}, & \\
 & \quad \varphi = \bigwedge \{\varphi_i \mid \mathsf{vars}(\varphi_i) \cap V \subseteq W\} & \\
 & \quad \psi = \bigwedge \{\varphi_i \mid \mathsf{vars}(\varphi_i) \cap V \not\subseteq W\} & \\
\mathsf{qi}_\exists(V, \varphi \vee \psi) & = \mathsf{qi}_\exists(V, \varphi) \vee \mathsf{qi}_\exists(V, \psi) & \\
\mathsf{qi}_\exists(V, \exists W.\varphi) & = \mathsf{qi}_\exists(V \cup W, \varphi) & \\
\mathsf{qi}_\exists(V, X(e)) & = \exists V \cap \mathsf{vars}(e).\, X(e) &
\end{array}
$$

In the remainder of this chapter, we regularly apply the quantifier-inside rewriter, both in definitions and examples. We remark that more advanced algorithms for partial quantifier elimination exist [53], although they are mostly focussed on quantified Boolean formulas.

7.3 Motivating Example

Consider the LPS (PDA, ϵ) that models a simple push down automaton. The linear process PDA is specified as follows.

$$
\begin{aligned}
&\mathit{PDA}(s{:}\mathit{List}(N)) = \\
&\qquad \sum_{n:N} (n \leq 50 \wedge n/3 = 5) \rightarrow \mathit{push}(n \bmod 6) \cdot \mathit{PDA}(n \triangleright s) \\
&\quad + (s \neq \epsilon) \rightarrow \mathit{pop}(\mathit{head}(s)) \cdot \mathit{PDA}(\mathit{tail}(s));
\end{aligned}
$$

In this linear process, we use the parameterised *List* data type to store a finite stack of natural numbers. The empty list is denoted ϵ and the functions *head* and *tail* respectively take the first element and the remainder of the list. Elements are prepended to the list with the operator $\triangleright$.

Note that this LPS has an infinite state space, since the list of values pushed onto the stack can grow arbitrarily long. Suppose we want to check the requirement that, starting from the initial state, no single value can be pushed onto the stack successively an infinite number of times. This is formalised by the μ-calculus formula

$\varphi = \forall m{:}N.\, \mu Y.\, [push(m)]Y$, which does not hold for our LPS (PDA, ϵ). When encoding the combination of PDA and φ into a PBES, we obtain the following:

$$\begin{aligned} \mu X(s{:}List(N)) &= \forall m{:}N.\, Y(s, m) \\ \mu Y(s{:}List(N), m{:}N) &= \forall n{:}N.\, (n \bmod 6 = m \wedge n \leq 50 \wedge n/3 = 5) \Rightarrow Y(n \triangleright s, m) \end{aligned}$$

This PBES cannot be solved through straightforward instantiation: due to the quantifier in the first equation, an instantiation that starts from $X(\epsilon)$ will not terminate. Moreover, the state space encoded in predicate variable Y is infinite. To resolve the latter problem, we apply parameter elimination to remove parameter s, which does not influence any conditions in the PBES. This yields the following PBES:

$$\begin{aligned} \mu X &= \forall m{:}N.\, Y(m) \\ \mu Y(m{:}N) &= \forall n{:}N.\, (n \bmod 6 = m \wedge n \leq 50 \wedge n/3 = 5) \Rightarrow Y(m) \end{aligned}$$

The quantifier over m is still problematic. Observe, however, that once a value has been fixed for m, it is never changed in the second equation. This might thus be a good place to apply the PBES substitution rule [59, Lemma 18], which allows replacing a PVI $Y(e)$ by $\varphi_Y[e/d_Y]$ in the right-hand side of X if and only if X occurs before Y in the PBES. Formally, the substitution rule claims that, for all environments η and δ, it holds that:

$$\begin{aligned} [\![(\sigma_X X(d{:}D) = \varphi_X)\mathcal{E}(\sigma_Y Y(d_Y{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta = \\ [\![(\sigma_X X(d{:}D) = \varphi_X[\varphi_Y/Y])\mathcal{E}(\sigma_Y Y(d_Y{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta \end{aligned}$$

By applying this to the right-hand side of Y we obtain the PBES

$$\begin{aligned} \mu X &= \forall m, n{:}N.\, (n \bmod 6 = m \wedge n \leq 50 \wedge n/3 = 5) \Rightarrow Y(m) \\ \mu Y(m{:}N) &= \forall n{:}N.\, (n \bmod 6 = m \wedge n \leq 50 \wedge n/3 = 5) \Rightarrow Y(m) \end{aligned}$$

This enables the use of other transformations, such as the one-point rewriter, which yields:

$$\begin{aligned} \mu X &= \forall n{:}N.\, (n \leq 50 \wedge n/3 = 5) \Rightarrow Y(n \bmod 6) \\ \mu Y(m{:}N) &= \forall n{:}N.\, (n \bmod 6 = m \wedge n \leq 50 \wedge n/3 = 5) \Rightarrow Y(m) \end{aligned}$$

And we find that the quantification over m has been eliminated! The remaining quantifiers can be finitely unfolded, which finally results in the following PBES:

$$\begin{aligned} \mu X &= Y(3) \wedge Y(4) \wedge Y(5) \\ \mu Y(m{:}N) &= (3 = m \Rightarrow Y(m)) \\ &\quad \wedge (4 = m \Rightarrow Y(m)) \\ &\quad \wedge (5 = m \Rightarrow Y(m)) \end{aligned}$$

This PBES can be solved easily with instantiation, since only five nodes in the corresponding parity game are reachable from X.

The example shows that the PBES substitution rule can be key to enabling other simplification techniques, and eventually solving a PBES. However, the approach is limited by the restriction that Y must occur after X. Moreover, there are no automated techniques to decide which and how many substitutions should be performed for the best results. It is also not possible to eliminate quantified variables that are not relevant by using the substitution rule, even when combined with parameter elimination. In the next section, we propose *quantifier propagation*, a technique that addresses some of these limitations.

7.4 Quantifier Propagation

Quantifier propagation circumvents the (need for) restrictions on the order of equations, such as those imposed by the substitution rule. The basic idea is best introduced with an example.

Example 7.1. Consider the following PBES with two equations:

$$\begin{aligned} \mu X &= \exists n{:}N.\, Y(2 + n', \mathit{true}) \\ \nu Y(n{:}N, b{:}B) &= (b \Rightarrow n \leq 1) \wedge X \end{aligned}$$

Since $Y(2 + n', \mathit{true})$ is the only PVI for Y, we can deduce that b always gets the value *true* in the right-hand side of Y. We can thus *propagate* this value for b, and we obtain the equivalent PBES

$$\begin{aligned} \mu X &= \exists n{:}N.\, Y(2 + n') \\ \nu Y(n{:}N) &= (\mathit{true} \Rightarrow n \leq 1) \wedge X \end{aligned}$$

This idea can be extended to quantifiers. Observe that $\exists n{:}N.\, Y(2 + n')$ is the only QPVI for Y; parameter n of Y consequently always has the value $2 + n'$, where n' is existentially bound. We also propagate the quantifier, resulting in

$$\begin{aligned} \mu X &= Y \\ \nu Y &= \exists n{:}N.\, (\mathit{true} \Rightarrow 2 + n' \leq 1) \wedge X \end{aligned}$$

Since $2 + n' \leq 1$ is unsatisfiable, the right-hand side of Y now reduces to *false*. This is a significant simplification over the original PBES. □

In the example, we benefited from the fact that the predicate variable Y only occurred in one QPVI. In the remainder of this section, we formalise the ideas behind quantifier propagation and show how it can be applied to predicate variables that occur in multiple QPVIs. To reason about subsequences of the quantifiers $\mathbf{Q}$ and the values $\mathbf{e}$ that occur in a QPVI $\mathbf{Q}.\, X(\mathbf{e})$, we introduce the concept of propagated quantified values, short *propagated values*.

Definition 7.2. A *propagated value* is a tuple $(\mathbf{Q}_p, \mathbf{d}_p, \mathbf{e}_p)$, where $\mathbf{Q}_p$ is a sequence of quantified variables, $\mathbf{d}_p$ is a sequence of variables and $\mathbf{e}_p$ is a sequence of expressions and $\mathbf{d}_p$ and $\mathbf{e}_p$ are of equal length and sort.

In the PBES manipulations we present below, we apply the information contained in propagated values to manipulate one or more right-hand sides in a PBES. We do this as follows: given a propagated value $(\mathbf{Q}_p, \mathbf{d}_p, \mathbf{e}_p)$ and a right-hand side φ, we substitute each variable $\mathbf{d}_p[i]$ by the corresponding expression $\mathbf{e}_p[i]$ and ensure that variables in $\mathbf{e}_p$ are bound according to $\mathbf{Q}_p$. This yields the updated right-hand side $\mathbf{Q}_p.\, \varphi[\mathbf{e}_p/\mathbf{d}_p]$. The substitution of $\mathbf{d}_p$ by $\mathbf{e}_p$ motivates the alternative notation $\mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$, which we will use henceforth. The propagated value $(\epsilon, \epsilon, \epsilon)$ is denoted $\bot$.

Example 7.3. Recall again the three PBESs from Example 7.1. Between the first and second PBES, we performed propagation with the propagated value $b := \mathit{true}$. The propagated value that characterises the transformation of the first PBES to the third PBES is $\exists n'{:}N.\, (n, b) := (2 + n', \mathit{true})$. □

In general, the modification of right-hand sides based on arbitrary propagated values does not preserve the semantics of the enclosing PBES. Whether transforming φ_X into $\mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p]$ is solution-preserving depends, among other things, on which QPVIs the corresponding predicate variable X occurs in. We capture this in the concept of a *propagation relation*.

Definition 7.4. A binary relation P between QPVIs and propagated values is a *propagation relation* if and only if for all $(\mathbf{Q}.\, X(\mathbf{e}))\ P\ \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$ the following conditions hold:

- $\mathbf{Q} = \mathbf{Q}' +\!\!+ \mathbf{Q}_p$ for some $\mathbf{Q}'$; and
- there is a characterising function f such that $\mathbf{e}_p \sqsubseteq_f \mathbf{e}$ and $\mathbf{d}_p \sqsubseteq_f \mathbf{d}_X$; and
- $\mathsf{vars}(\mathbf{e} \setminus \mathbf{e}_p) \cap \mathsf{v}(\mathbf{Q}_p) = \emptyset$ and $\mathsf{vars}(\mathbf{e}_p) \subseteq \mathsf{v}(\mathbf{Q}_p)$.

The largest propagation relation is $\hookrightarrow = \bigcup\{P \mid P \text{ is a propagation relation}\}$.

The largest propagation relation $\hookrightarrow$ is well defined, since the union of all propagation relations is itself a propagation relation. Note that every element in $\mathbf{d}_X$ is unique, so for every sequence $\mathbf{d}_p$, there is at most one function f such that $\mathbf{d}_p \sqsubseteq_f \mathbf{d}_X$. Thus, given $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$, the sequence $\mathbf{d}_p$ uniquely determines $\mathbf{e}_p$.

Let $\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$ denote left projection, *i.e.*, $\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow = \{y \mid \mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow y\}$. A propagated value $y = \mathbf{Q}'.\, \mathbf{d} := \mathbf{e}$ is associated to a predicate variable X iff $y \in \bigcup_{\mathbf{Q},\mathbf{e}} \mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$. To indicate this, we write $\mathbf{Q}'.\, \mathbf{d} :\overset{X}{=} \mathbf{e}$. A propagated value may be associated to more than one predicate variable, *e.g.*, the propagated value $\bot$ is associated to all predicate variables. Given some predicate variable X and two propagated values $y = \mathbf{Q}_p.\, \mathbf{d}_p :\overset{X}{=} \mathbf{d}_p$ and $y' = \mathbf{Q}'_p.\, \mathbf{d}'_p :\overset{X}{=} \mathbf{e}'_p$, we write $y \preccurlyeq_X y'$ if and only if, for some f, $\mathbf{d}_p \sqsubseteq_f \mathbf{d}'_p$, $\mathbf{e}_p \sqsubseteq_f \mathbf{e}'_p$ and $\mathbf{Q}_p$ is a suffix of $\mathbf{Q}'_p$. Note again

that there is at most one such f, due to the uniqueness of elements in $\mathbf{d}_X$. The preorder $\preccurlyeq_X$ gives rise to a unique infimum and supremum and $\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$ is finite, so $(\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow, \preccurlyeq_X)$ is a complete lattice. Although the correctness of our theory does not depend on choosing a specific propagated value in $\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$, we are often interested in the supremum, denoted with $\bigvee \mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$. In the next section, we also consider the complete lattice $(\mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow \cap\, \mathbf{Q}'.\, X(\mathbf{e}')\!\hookrightarrow, \preccurlyeq_X)$ that contains the propagated values of two QPVIs.

Example 7.5. Let X be a predicate variable, with parameters $\mathbf{d}_X = (d_1, d_2, d_3)$. We have $\forall m{:}N.\, \exists n{:}N.\, X(d_3, m + d_2, 2n)\!\hookrightarrow\, = \{\bot, \exists n{:}N.\, d_3 := 2n\}$. The parameters d_1 and d_2 do not occur in a propagated value since the corresponding expressions d_3 and $m + d_2$ respectively contain variables d_3 and d_2 that are not bound in the surrounding quantifiers, and hence violate the condition $\mathsf{vars}(\mathbf{e}_p) \subseteq \mathsf{v}(\mathbf{Q}_p)$. Consequently, $\mathsf{vars}(\mathbf{e} \setminus \mathbf{e}_p)$ contains at least m, d_2 and d_3; the quantifier $\forall m$ thus cannot occur in a propagated value since this would violate the condition $\mathsf{vars}(\mathbf{e} \setminus \mathbf{e}_p) \cap \mathsf{v}(\mathbf{Q}_p) = \emptyset$. For the QPVI $\varphi = \exists n{:}N.\, X(3, d_2, n + 1)$, we have $\bigvee \varphi\!\hookrightarrow\, = \exists n{:}N.\, d_1, d_3 := 3, n + 1$. □

With the conditions captured in $\hookrightarrow$, we are able to formalise the operation of quantifier propagation. To support propagation on a predicate variables that occur in multiple PVIs, quantifier propagation duplicates the corresponding equation.

Definition 7.6. Given a PBES $\mathcal{E} = \mathcal{E}_1(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X)\mathcal{E}_2$ and a QPVI $\mathbf{Q}.\, X(\mathbf{e})$, the function qprop applies *quantifier propagation* in the following way:

$$\begin{aligned}\mathsf{qprop}(\mathcal{E}_1(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X)\mathcal{E}_2, \mathbf{Q}.\, X(\mathbf{e})) = {} & \mathcal{E}_1[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})] \\ & (\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]) \\ & (\sigma \tilde{X}(\mathbf{d}_X{:}\mathbf{D}_X) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p]) \\ & \mathcal{E}_2[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\end{aligned}$$

where $\tilde{X} \notin \mathsf{bnd}(\mathcal{E})$ and $(\mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p) = \bigvee \mathbf{Q}.\, X(\mathbf{e})\!\hookrightarrow$.

In the resulting PBES, we created a new equation for $\tilde{X}$, whose right-hand side is constructed based on the right-hand side of X and the propagated value $\mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$. The QPVI $\mathbf{Q}.\, X(\mathbf{e})$ is replaced by $\mathbf{Q}.\, \tilde{X}(\mathbf{e})$ in the whole PBES, except for the right-hand side of $\tilde{X}$. Note that the variables in $\mathbf{d}_p$, which is a subsequence of the parameter list $\mathbf{d}_X$, do not occur in $\mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p]$; they are thus not relevant in $\tilde{X}$. This is also illustrated by the following example.

Example 7.7. Consider the PBES below.

$$\begin{aligned}\nu X(n, m{:}N) = m \leq n^2 \Rightarrow (n \geq 2 \wedge X(n - 1, m) \wedge Y) \\ \mu Y = X(0, 0) \vee \forall m'{:}N.\, X(4, m')\end{aligned}$$

We apply quantifier propagation based on the QPVI $\forall m'{:}N.\, X(4, m')$, whose maximal

propagated value is $\forall m'{:}N.\,(n,m) := (4,m')$. The resulting PBES is:

$$\begin{aligned}\nu X(n,m{:}N) &= m \leq n^2 \Rightarrow (n \geq 2 \wedge X(n-1,m) \wedge Y)\\ \nu \tilde{X}(n,m{:}N) &= \forall m'{:}N.\,(m' \leq 4^2 \Rightarrow (4 \geq 2 \wedge X(4-1,m') \wedge Y))\\ \mu Y &= X(0,0) \vee \forall m'{:}N.\,\tilde{X}(4,m')\end{aligned}$$

Neither parameter n, nor m of $\tilde{X}$ is relevant in the right-hand side: they can be removed through parameter elimination. Subsequently, the quantifier in the QPVI $\forall m'{:}N.\,\tilde{X}$ in the right-hand side of Y can be eliminated with the quantifier-inside rewriter. This results in the PBES

$$\begin{aligned}\nu X(n,m{:}N) &= m \leq n^2 \Rightarrow (n \geq 2 \wedge X(n-1,m) \wedge Y)\\ \nu \tilde{X} &= \forall m'{:}N.\,(m' \leq 4^2 \Rightarrow (4 \geq 2 \wedge X(4-1,m') \wedge Y))\\ \mu Y &= X(0,0) \vee \tilde{X}\end{aligned}$$

This PBES can more easily be solved through instantiation since parameter m of X is now bounded by 4^2. Note that it is not possible to obtain the same result when only using the substitution rule, since the equation for X occurs before the equation for Y in the PBES. □

Remark 7.8. An alternative approach to quantifier propagation, is allowing the propagation of free variables. For example, if X is a predicate variable with one parameter $b{:}B$, then the QPVI $\forall n{:}N.\,X(m+n \geq 12)$ can be replaced by $\tilde{X}(m)$, where the equation for $\tilde{X}$ is $\sigma_X \tilde{X}(m{:}N) = \forall n{:}N.\,\varphi_X[(m+n \geq 12)/b]$. Here, m is the free variable that is propagated as a parameter. Note that, as a result of introducing $\tilde{X}(m{:}N)$, the signature $\mathsf{sig}(\mathcal{E}')$ of the resulting PBES $\mathcal{E}'$ is no longer finite. Thus, it can be harder to solve than the original PBES. □

Before we prove that quantifier propagation preserves the semantics of the PBES, we reproduce several existing results from literature. These results will be used in the main correctness proof. Firstly, we rely on the substitution rule [59, Lemma 18] (see also Section 7.3); recall that it states that for all η and δ, it holds that

$$\begin{aligned}[\![(\sigma_X X(d{:}D) = \varphi_X)\mathcal{E}(\sigma_Y Y(d_Y{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta = \\ [\![(\sigma_X X(d{:}D) = \varphi_X[\varphi_Y/Y])\mathcal{E}(\sigma_Y Y(d_Y{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta\end{aligned}$$

Furthermore, we reproduce a number of auxiliary lemmata that occurred in [117] for the more general setting of fixpoint equation systems. Here, we present those lemmata in terms of PBESs.

These results also apply to PBESs that are not closed in the predicate variables. In particular, some proofs apply induction on the structure of a PBES, so the equation systems we consider there are, in many cases, not closed in the predicate variables. Furthermore, although we present the results for PBESs where each equation has a single parameter, each lemma can be trivially generalised to multi-parameter PBESs.

The first lemma states that a predicate environment that is used to interpret semantics of a PBES $\mathcal{E}$ is not changed in those variables that are not bound in $\mathcal{E}$.

Lemma 7.9. *[117, Lemma 11], [114, Lemma 1] Let $\mathcal{E}$ be a PBES. If $X \notin \mathsf{bnd}(\mathcal{E})$, then $([\![\mathcal{E}]\!]\eta\delta)(X) = \eta(X)$ for all δ.*

Proof. By structural induction on $\mathcal{E}$. For the base case, where $\mathcal{E} = \emptyset$, $([\![\emptyset]\!]\eta\delta)(X) = \eta(X)$ follows directly from the definition of the semantics of a PBES. For the induction step, where $\mathcal{E} = (\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}'$ for some $Y \neq X$ and $\mathcal{E}'$ such that $X \notin \mathsf{bnd}(\mathcal{E}')$, take as hypothesis that $([\![\mathcal{E}']\!]\eta\delta)(X) = \eta(X)$. We follow the reasoning below; application of the induction hypothesis is indicated with †.

$$\begin{aligned} & ([\![(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta)(X) \\ &= ([\![\mathcal{E}']\!]\eta[\sigma T_Y/Y]\delta)(X) \\ &\overset{(\dagger)}{=} \eta[\sigma T_Y/Y](X) \\ &= \eta(X) \end{aligned}$$

□

Next, we show how the substitution rule can be generalised to also allow substitution of X in its own right-hand side. We will refer to this as the *self-substitution rule*. We remark that this rule bears similarities to the *square rule* for fixpoints [117], which is defined as $\sigma x.\, f(x) = \sigma x.\, f(f(x))$.

Lemma 7.10. *[117, Lemma 15] For all η and δ, it holds that*

$$[\![(\sigma X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta = [\![(\sigma X(d{:}D) = \varphi_X[\varphi_X/X])\mathcal{E}]\!]\eta\delta$$

Proof. Consider the following predicate transformers, where $X \notin \mathsf{bnd}(\mathcal{E})$:

$$\begin{aligned} F(R) &= \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}]\!]\eta[R/X]\delta)\delta[v/d]\} \\ G(R) &= \{v \in \mathbb{D} \mid [\![\varphi_X[\varphi_X/X]]\!]([\![\mathcal{E}]\!]\eta[R/X]\delta)\delta[v/d]\} \\ H(R,P) &= \{v \in \mathbb{D} \mid [\![\varphi_X]\!](([\![\mathcal{E}]\!]\eta[R/X]\delta)[P/X])\delta[v/d]\} \end{aligned}$$

Below, we use the property $\sigma x.\, H(x,x) = \sigma x.\, H(x, H(x,x))$, known as the *unfolding rule*, which holds since H is monotonic and $(2^{\mathbb{D}}, \subseteq)$ is a complete lattice. For the proof of this property, we refer to [117, Lemma 5.7]. Furthermore, since $X \notin \mathsf{bnd}(\mathcal{E})$, we can use Lemma 7.9 to obtain:

$$[\![\mathcal{E}]\!]\eta[R/X]\delta = ([\![\mathcal{E}]\!]\eta[R/X]\delta)[R/X] \qquad (*)$$

As a consequence of the above, we have $F(R) = H(R,R)$. We derive equality of the

fixpoints σF and σG as follows:

$$\begin{aligned}
& \sigma R.\, F(R) \\
\overset{(*)}{=}\ & \sigma R.\, H(R, R) \\
=\ & \sigma R.\, H(R, H(R, R)) \\
\overset{(*)}{=}\ & \sigma R.\, H(R, F(R)) \\
\overset{\text{def.}H}{=}\ & \sigma R.\, \{v \in \mathbb{D} \mid [\![\varphi_X]\!](([\![\mathcal{E}]\!]\eta[R/X]\delta)[F(R)/X])\delta[v/d]\} \\
\overset{\text{def.}F}{=}\ & \sigma R.\, \{v \in \mathbb{D} \mid [\![\varphi_X]\!] \\
& \qquad (([\![\mathcal{E}]\!]\eta[R/X]\delta)[\{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}]\!]\eta[R/X]\delta)\delta[v/d]\}/X])\delta[v/d]\} \\
=\ & \sigma R.\, \{v \in \mathbb{D} \mid [\![\varphi_X[\varphi_X/X]]\!]([\![\mathcal{E}]\!]\eta[R/X]\delta)\delta[v/d]\} \\
=\ & \sigma R.\, G(R)
\end{aligned}$$

At the penultimate step, we moved the substitution of X from the predicate environment into the syntax. As a result of $\sigma F = \sigma G$, we have the following equality:

$$\begin{aligned}
& [\![(\sigma X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta \\
=\ & [\![\mathcal{E}]\!]\eta[\sigma F/X]\delta \\
=\ & [\![\mathcal{E}]\!]\eta[\sigma G/X]\delta \\
=\ & [\![(\sigma X(d{:}D) = \varphi_X[\varphi_X/X])\mathcal{E}]\!]\eta\delta
\end{aligned}$$ □

Now we are ready to prove the correctness of quantifier propagation through several lemmata and a theorem. The first lemma provides the core argument for why all QPVIs $\mathbf{Q}.\, X(\mathbf{e})$ may be replaced by $\mathbf{Q}.\, \tilde{X}(\mathbf{e})$, assuming that the predicate environment these QPVIs are evaluated under respects the equations for X and $\tilde{X}$.

Lemma 7.11. *Let η be a predicate environment such that for all δ', it holds that there is some η' such that*

$$[\![X(\mathbf{e})]\!]\eta\delta' = [\![\varphi_X]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X] \text{ and } [\![\tilde{X}(\mathbf{e})]\!]\eta\delta' = [\![\mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X] \tag{†}$$

Then, for all QPVIs $\mathbf{Q}.\, X(\mathbf{e})$ such that $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$ and all δ, it holds that $[\![\mathbf{Q}.\, X(\mathbf{e})]\!]\eta\delta = [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]\eta\delta$.

Proof. Let η be as above and let $\mathbf{Q}.\, X(\mathbf{e})$ be an arbitrary QPVI such that $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$. Furthermore, let δ be arbitrary and assume that the environment for which (†) holds is η'. We perform the following derivation. Below, let $\delta' = \delta[\mathsf{v}(\boldsymbol{\mathcal{Q}})/\mathsf{v}(\mathbf{Q})]$.

$$\begin{aligned}
& [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]\eta\delta \\
=\ & \boldsymbol{\mathcal{Q}}.\, [\![\tilde{X}(\mathbf{e})]\!]\eta\delta' \\
\overset{(\dagger)}{=}\ & \boldsymbol{\mathcal{Q}}.\, [\![\mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X] \\
=\ & \boldsymbol{\mathcal{Q}}.\, \boldsymbol{\mathcal{Q}}_p.\, [\![\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X][\mathsf{v}(\boldsymbol{\mathcal{Q}}_p)/\mathsf{v}(\mathbf{Q}_p)]
\end{aligned}$$

By $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$, we have $\mathsf{vars}(\mathbf{e} \setminus \mathbf{e}_p) \cap \mathsf{v}(\mathbf{Q}_p) = \emptyset$ and $\mathsf{vars}(\mathbf{e}_p) \subseteq \mathsf{v}(\mathbf{Q}_p)$, *i.e.*, the variables in $\mathbf{e}_p$ only occur in $\mathsf{v}(\mathbf{Q}_p)$ and vice versa. Hence, $\varphi_X[\mathbf{e}_p/\mathbf{d}_p]$ can be eliminated as follows:

$$\begin{aligned}
&\mathcal{Q}.\, \mathcal{Q}_p.\, [\![\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X][\mathsf{v}(\mathcal{Q}_p)/\mathsf{v}(\mathbf{Q}_p)] \\
&= \mathcal{Q}.\, \mathcal{Q}_p.\, [\![\varphi_X]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X][[\![\mathbf{e}_p]\!]\delta_0[\mathsf{v}(\mathcal{Q}_p)/\mathsf{v}(\mathbf{Q}_p)]/\mathbf{d}_p]
\end{aligned}$$

Since $\mathcal{Q}_p$ is a suffix of $\mathcal{Q}$, those values in $\mathcal{Q}$ are bound again in $\mathcal{Q}_p$. Those irrelevant values can be eliminated. Furthermore, we have $\mathbf{e}_p \sqsubseteq \mathbf{e}$, so we can merge the data environment updates:

$$\begin{aligned}
&\mathcal{Q}.\, \mathcal{Q}_p.\, [\![\varphi_X]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X][[\![\mathbf{e}_p]\!]\delta_0[\mathsf{v}(\mathcal{Q}_p)/\mathsf{v}(\mathbf{Q}_p)]/\mathbf{d}_p] \\
&= \mathcal{Q}.\, [\![\varphi_X]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_X] \\
&\overset{(\dagger)}{=} \mathcal{Q}.\, [\![X(\mathbf{e})]\!]\eta\delta' \\
&= [\![\mathbf{Q}.\, X(\mathbf{e})]\!]\eta\delta
\end{aligned}$$

□

As per the following lemma, a predicate environment η that identifies $\mathbf{Q}.\, X(\mathbf{e})$ and $\mathbf{Q}.\, \tilde{X}(\mathbf{e})$, allows the former to be substituted by the latter in a PBES $\mathcal{E}$, without affecting its semantics under η.

Lemma 7.12. *Let $\mathcal{E}$ be a PBES such that $X, \tilde{X} \notin \mathsf{bnd}(\mathcal{E})$ and η a predicate environment. If, for all δ, $[\![\mathbf{Q}.\, X(\mathbf{e})]\!]\eta\delta = [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]\eta\delta$, then it holds, for all δ, that $[\![\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\eta\delta$.*

Proof. By induction. Let $\mathcal{E}$ and η be as above and let δ be arbitrary. For the base case, where $\mathcal{E} = \emptyset$, the lemma trivially holds. For the induction step, we assume that $[\![\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\eta\delta$ and we show that the same holds when an equation $\sigma Y(\mathbf{d}{:}\mathbf{D}) = \varphi_Y$, with $Y \neq X$ and $Y \neq \tilde{X}$, is added in front of $\mathcal{E}$. Consider the following three predicate transformers:

$$\begin{aligned}
T_Y(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_Y]\!]([\![\mathcal{E}]\!]\eta[R/Y]\delta)\delta[\mathbf{v}/\mathbf{d}]\} \\
T'_Y(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_Y]\!]([\![\mathcal{E}[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\eta[R/Y]\delta)\delta[\mathbf{v}/\mathbf{d}]\} \\
T''_Y(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_Y[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]([\![\mathcal{E}[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\eta[R/Y]\delta)\delta[\mathbf{v}/\mathbf{d}]\}
\end{aligned}$$

Observe that, for all R, $T_Y(R) = T'_Y(R)$ by the induction hypothesis and the fact that the values for X and $\tilde{X}$ are not overwritten in $\eta[R/Y]$. Furthermore, we have the assumption that $[\![\mathbf{Q}.\, X(\mathbf{e})]\!]\eta\delta = [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]\eta\delta$ and Lemma 7.9 yields

$$([\![\mathcal{E}[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\eta[R/Y]\delta)(X) = \eta[R/Y](X) = \eta(X)$$

(and likewise for $\tilde{X}$), so the substitution can also be applied to φ_Y, and we obtain $T'_Y(R) = T''_Y(R)$. Hence these transformers all have the same fixpoints, and we derive

the following:

$$\begin{aligned}
&[\![(\sigma Y(\mathbf{d}{:}\mathbf{D}) = \varphi_Y)\mathcal{E}]\!]\eta\delta \\
&= [\![\mathcal{E}]\!]\eta[\sigma T_Y/Y]\delta \\
&\overset{(IH)}{=} [\![\mathcal{E}[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]\eta[\sigma T_Y/Y]\delta \\
&\overset{(IH)}{=} [\![\mathcal{E}[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]\eta[\sigma T'_Y/Y]\delta \\
&= [\![\mathcal{E}[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]\eta[\sigma T''_Y/Y]\delta \\
&= [\![(\sigma Y(\mathbf{d}{:}\mathbf{D}) = \varphi_Y[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})])\mathcal{E}[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]\eta\delta
\end{aligned}$$

□

We next combine the results of the previous two lemmata and show how they can be applied to show correctness of the substitution $\mathcal{E}_2[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]$ in the context of quantifier propagation. Before we proceed, however, we shortly introduce the application of Bekič's lemma on simultaneous fixpoints [13] on PBES semantics. The standard definition of PBES semantics recurses through the PBES one equation at a time. For consecutive equations that carry the same fixpoint symbol, it is also possible to capture both their semantics in the fixpoint of one single transformer, as follows:

$$[\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}]\!]\eta[\sigma T_{X,Y}/(X,Y)]\delta$$

where

$$\begin{aligned}
T_{X,Y}(R,R') = (&\{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}]\!]\eta[(R,R')/(X,Y)]\delta)\delta[v/d]\}, \\
&\{v \in \mathbb{D} \mid [\![\varphi_Y]\!]([\![\mathcal{E}]\!]\eta[(R,R')/(X,Y)]\delta)\delta[v/d]\})
\end{aligned}$$

Here, we restricted ourselves to the case of two consecutive equations, since this suffices for our application.

Lemma 7.13. *Let $\mathcal{E} = (\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma\tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2$ be a PBES and $\mathbf{Q}.\,X(\mathbf{e})$ a QPVI such that $\mathbf{Q}.\,X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\,\mathbf{d}_p := \mathbf{e}_p$. Then, for all η and δ, it holds that*

$$[\![\mathcal{E}]\!]\eta\delta = [\![(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma\tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]\eta\delta$$

Proof. Let $\mathcal{E}$ and $\mathbf{Q}.\,X(\mathbf{e})$ be as above and let η and δ be arbitrary. Furthermore, let $\mathcal{E}'_2 = \mathcal{E}_2[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]$. We define the following two predicate transformers relative to $\mathcal{E}_2$ and $\mathcal{E}'_2$:

$$\begin{aligned}
T_{X,\tilde{X}}(R,\tilde{R}) = (&\{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}_2]\!]\eta[(R,\tilde{R})/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}, \\
&\{\mathbf{v} \in \mathbb{D} \mid [\![\mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]([\![\mathcal{E}_2]\!]\eta[(R,\tilde{R})/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}) \\
T'_{X,\tilde{X}}(R,\tilde{R}) = (&\{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}'_2]\!]\eta[(R,\tilde{R})/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}, \\
&\{\mathbf{v} \in \mathbb{D} \mid [\![\mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]([\![\mathcal{E}'_2]\!]\eta[(R,\tilde{R})/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\})
\end{aligned}$$

To show that $\sigma T_{X,\tilde{X}} = \sigma T'_{X,\tilde{X}}$, we prove that $T_{X,\tilde{X}}$ and $T'_{X,\tilde{X}}$ have the same fixpoints. Let $A \in 2^{\mathbb{D}} \times 2^{\mathbb{D}}$ be an arbitrary fixpoint of $T_{X,\tilde{X}}$. We derive:

$$\begin{aligned}
&[\![X(\mathbf{e})]\!]\eta[A/(X,\tilde{X})]\delta \\
&= [\![\mathbf{e}]\!]\delta \in \eta[A/(X,\tilde{X})](X) \\
&= [\![\mathbf{e}]\!]\delta \in A[1] \\
&= [\![\mathbf{e}]\!]\delta \in T_{X,\tilde{X}}(A)[1] \\
&= [\![\mathbf{e}]\!]\delta \in \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\} \\
&= [\![\varphi_X]\!]([\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[[\![\mathbf{e}]\!]\delta/d]
\end{aligned}$$

Similarly, we derive that

$$[\![\tilde{X}(\mathbf{e})]\!]\eta[A/(X,\tilde{X})]\delta = [\![\mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]([\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[[\![\mathbf{e}]\!]\delta/d]$$

Hence, $\eta[A/(X,\tilde{X})]$ satisfies the assumptions of Lemma 7.11, and consequently, also those of Lemma 7.12. We obtain $[\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta = [\![\mathcal{E}'_2]\!]\eta[A/(X,\tilde{X})]\delta$. With this, we can reason about A as follows:

$$\begin{aligned}
&A \\
&= T_{X,\tilde{X}}(A) \\
&= (\{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}, \\
&\quad \{\mathbf{v} \in \mathbb{D} \mid [\![\mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]([\![\mathcal{E}_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}) \\
&= (\{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}'_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}, \\
&\quad \{\mathbf{v} \in \mathbb{D} \mid [\![\mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p]]\!]([\![\mathcal{E}'_2]\!]\eta[A/(X,\tilde{X})]\delta)\delta[\mathbf{v}/\mathbf{d}]\}) \\
&= T'_{X,\tilde{X}}(A)
\end{aligned}$$

Thus, A is also a fixpoint of $T'_{X,\tilde{X}}$. An analogous reasoning can be followed to show that every fixpoint of $T'_{X,\tilde{X}}$ is also a fixpoint of $T_{X,\tilde{X}}$. We conclude that $\sigma T_{X,\tilde{X}} = \sigma T'_{X,\tilde{X}}$.

As explained earlier, with Bekič's lemma on simultaneous fixpoints we have $[\![\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}_2]\!]\eta[\sigma T_{X,\tilde{X}}/(X,\tilde{X})]\delta$. Furthermore, as a special case of the reasoning above, it holds that $[\![\mathcal{E}_2]\!]\eta[\sigma T_{X,\tilde{X}}/(X,\tilde{X})]\delta = [\![\mathcal{E}'_2]\!]\eta[\sigma T_{X,\tilde{X}}/(X,\tilde{X})]\delta$. We subsequently apply these two facts in the next derivation:

$$\begin{aligned}
&[\![\mathcal{E}]\!]\eta\delta \\
&= [\![\mathcal{E}_2]\!]\eta[\sigma T_{X,\tilde{X}}/(X,\tilde{X})]\delta \\
&= [\![\mathcal{E}'_2]\!]\eta[\sigma T_{X,\tilde{X}}/(X,\tilde{X})]\delta \\
&= [\![\mathcal{E}'_2]\!]\eta[\sigma T'_{X,\tilde{X}}/(X,\tilde{X})]\delta \\
&= [\![(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma\tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}'_2]\!]\eta\delta
\end{aligned}$$ □

The correctness of the other substitutions performed by the qprop function is relatively straightforward to prove through the use of the substitution rule.

Lemma 7.14. *Let* $\mathcal{E} = \mathcal{E}_1(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma \tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2$ *be a PBES and* $\mathbf{Q}.\, X(\mathbf{e})$ *a QPVI such that* $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$. *Furthermore let* $\mathcal{E}'$ *be defined as*

$$\mathcal{E}' = \mathcal{E}_1[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})](\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})])\\ (\sigma \tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2$$

Then, for all η *and* δ, *it holds that* $[\![\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}']\!]\eta\delta$.

Proof. Let the PBES $\mathcal{E}$ and the QPVI $\mathbf{Q}.\, X(\mathbf{e})$ be as above. Consider an equation $\sigma_Y Y(\mathbf{d}_Y{:}\mathbf{D}_Y) = \varphi_Y$ that occurs before $\tilde{X}$ in $\mathcal{E}$ ($Y = X$ is allowed). We show that the semantics of $\mathcal{E}$ are preserved if $\mathbf{Q}.\, X(\mathbf{e})$ is replaced by $\mathbf{Q}.\, \tilde{X}(\mathbf{e})$ in the right-hand side of Y. First, we apply the substitution rule (or, if $Y = X$, the self-substitution rule of Lemma 7.10) to obtain

$$\sigma_Y Y(\mathbf{d}_Y{:}\mathbf{D}_Y) = \varphi_Y[\mathbf{Q}.\, \varphi_X[\mathbf{e}/\mathbf{d}]/\mathbf{Q}.\, X(\mathbf{e})]$$

Then, we duplicate the quantifiers in $\mathbf{Q}_p$, which is a suffix of $\mathbf{Q}$, and we duplicate the substitution of $\mathbf{d}_p$, which is possible since $\mathbf{d}_p \sqsubseteq_f \mathbf{d}$ and $\mathbf{e}_p \sqsubseteq_f \mathbf{e}$ for some f. This yields:

$$\sigma_Y Y(\mathbf{d}_Y{:}\mathbf{D}_Y) = \varphi_Y[\mathbf{Q}.\, (\mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p])[\mathbf{e}/\mathbf{d}]/\mathbf{Q}.\, X(\mathbf{e})]$$

Now we can substitute back the PVI of $\tilde{X}$, resulting in:

$$\sigma_Y Y(\mathbf{d}_Y{:}\mathbf{D}_Y) = \varphi_Y[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]$$

The same procedure can be repeated for all equations that precede $\tilde{X}$ (including X), without affecting the semantics. This results in the desired PBES $\mathcal{E}'$ and we conclude that $[\![\mathcal{E}]\!]\eta\delta = [\![\mathcal{E}']\!]\eta\delta$ for all η and δ. □

Observe that, according the semantics of a PBES, if $[\![\mathcal{E}_2]\!]\eta\delta = [\![\mathcal{E}_2']\!]\eta\delta$ for all η and δ, then $[\![\mathcal{E}_1\mathcal{E}_2]\!]\eta\delta = [\![\mathcal{E}_1\mathcal{E}_2']\!]\eta\delta$ for all η and δ, as long as $\mathsf{bnd}(\mathcal{E}_1) \cap (\mathsf{bnd}(\mathcal{E}_2) \cup \mathsf{bnd}(\mathcal{E}_2')) = \emptyset$. This observation is formalised in [59, Corollary 16], albeit with slightly different assumptions on the relation between $\mathcal{E}_2$ and $\mathcal{E}_2'$. We apply the idea of compositional reasoning in the proof of the next theorem.

Theorem 7.15. *Let* $\mathcal{E}$ *be a closed PBES and* $\mathbf{Q}.\, X(\mathbf{e})$ *a QPVI that occurs in* $\mathcal{E}$. *Then, for all* $Y \in \mathsf{bnd}(\mathcal{E})$ *and* $\mathbf{v} \in \mathbb{D}$, *it holds that* $\mathbf{v} \in [\![\mathcal{E}]\!](Y)$ *iff* $\mathbf{v} \in [\![\mathsf{qprop}(\mathcal{E}, \mathbf{Q}.\, X(\mathbf{e}))]\!](Y)$.

Proof. Let $\mathcal{E}$ and $\mathbf{Q}.\, X(\mathbf{e})$ be as above and let $\bigvee \mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow = \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$. If $\mathcal{E} = \mathcal{E}_1(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X)\mathcal{E}_2$ for some $\mathcal{E}_1$ and $\mathcal{E}_2$, then let $\mathcal{E}'$ be the PBES defined as $\mathcal{E}' = \mathcal{E}_1(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X)(\sigma \tilde{X}(\mathbf{d}_X{:}\mathbf{D}_X) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2$. Clearly, we have for all $Y \in \mathsf{bnd}(\mathcal{E})$ and $\mathbf{v} \in \mathbb{D}$ that $\mathbf{v} \in [\![\mathcal{E}]\!](Y)$ iff $\mathbf{v} \in [\![\mathcal{E}']\!](Y)$, since the addition

of an equation does not influence the solution of the other predicate variables. By Lemma 7.13 and compositionality of PBESs, we have

$$[\![\mathcal{E}']\!] = [\![\mathcal{E}_1(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X) \\ (\sigma\tilde{X}(\mathbf{d}_X{:}\mathbf{D}_X) = \mathbf{Q}_p.\,\varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2[\mathbf{Q}.\,\tilde{X}(\mathbf{e})/\mathbf{Q}.\,X(\mathbf{e})]]\!]$$

On the latter PBES, we apply Lemma 7.14, which yields $[\![\mathcal{E}']\!] = [\![\mathsf{qprop}(\mathcal{E}, \mathbf{Q}.\,X(\mathbf{e}))]\!]$. We obtain that for all $Y \in \mathsf{bnd}(\mathcal{E})$ and $\mathbf{v} \in \mathbb{D}$, it holds that $\mathbf{v} \in [\![\mathcal{E}]\!](Y)$ iff $\mathbf{v} \in [\![\mathsf{qprop}(\mathcal{E}, \mathbf{Q}.\,X(\mathbf{e}))]\!](Y)$ □

Although Example 7.7 and the example from Section 7.3 illustrate that quantifier propagation works well for some PBESs, two issues still remain. First of all, if we want to create a fully automated implementation of quantifier propagation, we have to decide which QPVIs propagation should be applied on. Propagating all QPVIs can blow up the number of equations in the PBES unnecessarily. For example, in the PBES $\nu X(n{:}N) = X(0) \wedge X(1) \wedge X(2) \wedge X(3)$, propagation of all QPVIs introduces four new equations, without actually reducing the size of the underlying graph. It is thus desirable to determine *a priori* which QPVIs one can propagate without significantly increasing the size of the PBES itself.

The second issue is that, even if we have such a decision procedure, we are never able to eliminate a quantifier if its variable always occurs in a PVI, even after several propagation steps. This is illustrated in the next example.

Example 7.16. Consider the PBES below.

$$\mu X = \forall n'{:}N.\,Y(n') \\ \nu Y(n{:}N) = n \leq 5 \wedge Y(n)$$

If we apply quantifier propagation on $\forall n'{:}N.\,Y(n')$, we obtain a new equation $\nu\tilde{Y} = \forall n'{:}N.\,n' \leq 5 \wedge Y(n')$. After applying the quantifier-inside rewriter to the right-hand side of $\tilde{Y}$, its equation becomes $\nu\tilde{Y} = (\forall n'{:}N.\,n' \leq 5) \wedge (\forall n'{:}N.\,Y(n'))$, and we again encounter the QPVI $\forall n'{:}N.\,Y(n')$. Now, propagation is again possible on $\forall n'{:}N.\,Y(n')$, and we obtain another equation $\nu\tilde{Y}' = \forall n'{:}N.\,n' \leq 5 \wedge Y(n')$. Its right-hand side is the same as the original right-hand side of $\tilde{Y}$, and we have actually increased the size of the underlying dependency graph. Ideally, we would like to identify that the equations for $\tilde{Y}$ and $\tilde{Y}'$ are the same, and substitute $\forall n'{:}N.\,Y(n')$ by $\tilde{Y}$, yielding

$$\mu X = \tilde{Y} \\ \nu Y(n{:}N) = n \leq 5 \wedge Y(n) \\ \nu\tilde{Y} = (\forall n'{:}N.\,n' \leq 5) \wedge \tilde{Y}$$ □

We address these issues in the next section, where we propose a fully automated technique that can determine which propagated values can be eliminated.

7.5 Finding Global Propagated Values

To determine whether the propagation of QPVIs results in many new equations and to find quantifiers that can be eliminated according to the scheme suggested in Example 7.16, we gather all QPVIs that occur in a PBES and analyse their effect on quantifier propagation. The analysis starts from a target node $\hat{X}(\hat{\mathbf{e}})$ and finds, for each predicate variable $X \in \mathsf{bnd}(\mathcal{E})$, the largest propagated value (according to $\preccurlyeq_X$) that is related to all QPVIs of X which occur as a transitive dependency of $\hat{X}(\hat{\mathbf{e}})$. The resulting *global propagated values* can be used to transform the PBES, without the need to introduce new equations. We require that variables bound in quantifiers can be distinguished from parameters, so we henceforth assume that parameters are never bound in quantifiers.

To facilitate our static analysis, we extract the set of all QPVIs that occur in a predicate formula with the function qiocc:

$$\begin{aligned}
\mathsf{qiocc}(\mathbf{Q}, b) &= \emptyset \\
\mathsf{qiocc}(\mathbf{Q}, \neg\varphi) &= \mathsf{qiocc}(\epsilon, \varphi) \\
\mathsf{qiocc}(\mathbf{Q}, \varphi \wedge \psi) &= \mathsf{qiocc}(\epsilon, \varphi) \cup \mathsf{qiocc}(\epsilon, \psi) \\
\mathsf{qiocc}(\mathbf{Q}, \varphi \vee \psi) &= \mathsf{qiocc}(\epsilon, \varphi) \cup \mathsf{qiocc}(\epsilon, \psi) \\
\mathsf{qiocc}(\mathbf{Q}, \forall d{:}D.\, \varphi) &= \mathsf{qiocc}(\mathbf{Q} +\!\!+ [(\forall, d)], \varphi) \\
\mathsf{qiocc}(\mathbf{Q}, \exists d{:}D.\, \varphi) &= \mathsf{qiocc}(\mathbf{Q} +\!\!+ [(\exists, d)], \varphi) \\
\mathsf{qiocc}(\mathbf{Q}, X(\mathbf{e})) &= \{\mathbf{Q}.\, X(\mathbf{e})\}
\end{aligned}$$

where $\mathbf{Q}$ is a sequence of quantified variables. Furthermore, we overload qiocc, such that we have $\mathsf{qiocc}(\varphi) = \mathsf{qiocc}(\epsilon, \varphi)$.

We introduce a special value $\top$ such that $y \preccurlyeq_X \top$ for all propagated values y. The definition of the infimum $\wedge$ in the lattice $(\bigcup_{\mathbf{Q},\mathbf{e}} \mathbf{Q}.\, X(\mathbf{e})\hookrightarrow \cup \{\top\}, \preccurlyeq_X)$ follows in the standard way. To support the intuition, we give an equivalent definition of $\wedge$:

$$\begin{aligned}
\top \wedge \top &= \top \\
\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e} \wedge \top &= \mathbf{Q}.\, \mathbf{e} :\overset{X}{=} \mathbf{e} \\
\top \wedge \mathbf{Q}'.\, \mathbf{d}' :\overset{X}{=} \mathbf{e}' &= \mathbf{Q}'.\, \mathbf{e}' :\overset{X}{=} \mathbf{e}' \\
\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e} \wedge \mathbf{Q}'.\, \mathbf{d}' :\overset{X}{=} \mathbf{e}' &= \bigvee(\mathbf{Q}.\, X(\mathbf{d}_X[\mathbf{e}/\mathbf{d}])\hookrightarrow \cap\, \mathbf{Q}'.\, X(\mathbf{d}_X[\mathbf{e}'/\mathbf{d}'])\hookrightarrow)
\end{aligned}$$

The definition of the first three cases is straightforward. In the fourth case, where we compute the infimum of two propagated values, we do the following. First, note that, given a propagated value $\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e}$, a QPVI that contains exactly the same information is $\mathbf{Q}.\, X(\mathbf{d}_X[\mathbf{e}/\mathbf{d}])$ After all, we have $(\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e}) = \bigvee \mathbf{Q}.\, X(\mathbf{d}_X[\mathbf{e}/\mathbf{d}])\hookrightarrow$. To compute $\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e} \wedge \mathbf{Q}'.\, \mathbf{d}' :\overset{X}{=} \mathbf{e}'$, we take the largest (according to $\preccurlyeq_X$) propagated value that is related by $\hookrightarrow$ to both QPVIs $\mathbf{Q}.\, X(\mathbf{d}_X[\mathbf{e}/\mathbf{d}])$ and $\mathbf{Q}'.\, X(\mathbf{d}_X[\mathbf{e}'/\mathbf{d}'])$. More concretely, we compare $\mathbf{d}_X[\mathbf{e}/\mathbf{d}]$ and $\mathbf{d}_X[\mathbf{e}'/\mathbf{d}']$ at each position i. If they are different, the parameter $\mathbf{d}_X[i]$ should not occur in the resulting propagated value. The quantifiers in the resulting propagated values must originate from the innermost quantifiers in $\mathbf{Q}$ and $\mathbf{Q}'$. Furthermore, we also follow the restrictions on the variables

occurring in the quantifiers and the expressions. We lift $\wedge$ to sets in the ordinary way (note that $\wedge$ is commutative and associative); $\wedge$ of an empty set yields $\top$.

Example 7.17. Consider the following PBES:

$$\nu X(n{:}N) = \mathbf{Q}_1.\, X(\mathbf{e}_1) \wedge \mathbf{Q}_2.\, X(\mathbf{e}_2)$$
$$\mu Y(n_1, n_2, n_3, n_4, n_5{:}N) = X(n_1 + n_2 + n_3 + n_4 + n_5)$$

where

$$\mathbf{Q}_1.\, X(\mathbf{e}_1) = \forall m_1{:}N.\, \exists m_2{:}N.\, \forall m_3{:}N.\, \exists m_4{:}N.\, Y(n + m_1, m_2, m_3 + 2, m_3, m_4)$$
$$\mathbf{Q}_2.\, X(\mathbf{e}_2) = \forall m_1{:}N.\, \forall m_2{:}N.\, \forall m_3{:}N.\, \exists m_4{:}N.\, Y(n + m_1, m_2, 5m_3, m_3, m_4)$$

We have

$$(\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e}) = \bigvee(\mathbf{Q}_1.\, X(\mathbf{e}_1)\hookrightarrow \cap\, \mathbf{Q}_2.\, X(\mathbf{e}_2)\hookrightarrow) = (\exists m_4{:}N.\, n_5 := m_4)$$

The longest common suffix of $\mathbf{Q}_1$ and $\mathbf{Q}_2$ is $(\forall, m_3)(\exists, m_4)$. Thus, $n + m_1 \notin \mathbf{e}$ and $m_2 \notin \mathbf{e}$, by the condition $\mathsf{vars}(\mathbf{e}) \subseteq \mathsf{v}(\mathbf{Q})$. Furthermore, the expressions $m_3 + 2$ and $5m_3$ are not equivalent, so $n_3 \notin \mathbf{d}$. Consequently, $(\forall, m_3) \notin \mathbf{Q}$, to satisfy the condition $\mathsf{vars}(\mathbf{d}_X \setminus \mathbf{d}) \cap \mathsf{v}(\mathbf{Q}) = \emptyset$, and $m_3 \notin \mathbf{e}$, to satisfy $\mathsf{vars}(\mathbf{e}) \subseteq \mathsf{v}(\mathbf{Q})$. □

Similar to quantifier propagation, correctness of the approach we propose below does not depend on the choice of a propagated value in the definition of $\wedge$. However, the use of the supremum does ensure that we extract as much information as possible about possibly globally quantified parameters.

Let $\hat{X}(\hat{\mathbf{e}})$ be a target node for which we want to know the solution. Then, given a PBES $\mathcal{E}$, we can detect globally quantified parameters with the following algorithm. For now, assume that $\mathsf{guard}^{X(\mathbf{e})}(\varphi)$ always returns *true*; its function will be explained later.

$$\mathsf{pv}_0(\hat{X}) = (\mathbf{d}_X :\overset{X}{=} \hat{\mathbf{e}})$$
for every $X \in \mathsf{bnd}(\mathcal{E}) \setminus \{\hat{X}\}$,
$$\mathsf{pv}_0(X) = \top$$
for every $X \in \mathsf{bnd}(\mathcal{E})$,
$$\mathsf{pv}_{i+1}(X) = \bigwedge(\{\mathsf{pv}_i(X)\} \cup \bigcup\{\bigvee\mathbf{Q}'.\, X(\mathbf{e}')\hookrightarrow \mid \exists Y \in \mathsf{bnd}(\mathcal{E}).$$
$$\mathsf{pv}_i(Y) = (\mathbf{Q}.\, \mathbf{d} :\overset{Y}{=} \mathbf{e}) \wedge (\mathbf{Q}'.\, X(\mathbf{e}')) \in \mathsf{qiocc}(\mathsf{qi}(\mathbf{Q}.\, \varphi_Y[\mathbf{e}/\mathbf{d}])) \wedge$$
$$\exists \delta.\, [\![\mathsf{guard}^{X(\mathbf{e})}(\mathbf{Q}.\, \varphi_Y[\mathbf{e}/\mathbf{d}])]\!]\delta\})$$
$$\text{Output: } \mathsf{gpv} = \bigwedge_{i \geq 0} \mathsf{pv}_i$$

Initially, we take the ground terms from $\hat{\mathbf{e}}$ and construct a propagated value $\mathbf{d}_X :\overset{X}{=} \hat{\mathbf{e}}$. For other variables, the initial propagated value is set to $\top$. In every iteration, we

construct right-hand sides based on the propagated values we have found so far, in a similar fashion as quantifier propagation. From the new right-hand sides, we extract quantified predicates and compute their infimum with the current propagated value. This propagation of information continues until we reach a fixpoint gpv. We remark that this algorithm generalises the algorithm for finding constant parameters [106], since any constant value computed by the latter algorithm is also found by our algorithm.

Applying the global propagated value Given a PBES $\mathcal{E}$, a target node $\hat{X}(\hat{\mathbf{e}})$ and the global propagated value function gpv, we rewrite the right-hand side of each equation by traversing the PBES recursively:

$$\begin{aligned}
&\mathsf{global\text{-}prop}(\emptyset) = \emptyset\\
&\mathsf{global\text{-}prop}((\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \varphi_X)\mathcal{E}') =\\
&\qquad\begin{cases}(\sigma X(\mathbf{d}_X{:}\mathbf{D}_X) = \mathbf{Q}.\,\varphi_X[\mathbf{e}/\mathbf{d}])\mathsf{global\text{-}prop}(\mathcal{E}') & \text{if } \mathsf{gpv}(X) = \mathbf{Q}.\,\mathbf{d} :\overset{X}{=} \mathbf{e}\\ \mathsf{global\text{-}prop}(\mathcal{E}') & \text{if } \mathsf{gpv}(X) = \top\end{cases}
\end{aligned}$$

Note that the parameters of $\mathbf{d}_X$ that are also contained in $\mathbf{d}$ do not occur any longer in the right-hand side of X. Those parameters are now redundant, and they can be eliminated with the automatic techniques described in Section 7.1. Subsequently, the quantifier-inside rewriter can eliminate quantifiers from QPVIs where parameter elimination has reduced the set of free variables. If $\mathsf{gpv}(X) = \top$, then the equation for X is unreachable from $\hat{X}(\hat{\mathbf{e}})$, and it is removed from $\mathcal{E}$.

Example 7.18. Consider the target node $X(4)$ and the PBES below:

$$\begin{aligned}
\mu X(n{:}N) &= \forall m'{:}N.\, Y(n, m')\\
\nu Y(n, m{:}N) &= Z(n, m) \wedge (n \geq 2 \Rightarrow Y(n/2, m)) \wedge (n \leq 5 \Rightarrow Y(n + 7, m))\\
\mu Z(n, m{:}N) &= (n < m \wedge Z(n, m - 1)) \vee (n \geq m)
\end{aligned}$$

The globally propagated values relative to $X(4)$ are iteratively computed as follows:

$$\begin{array}{lll}
\mathsf{pv}_0(X) = n := 4 & \mathsf{pv}_0(Y) = \top & \mathsf{pv}_0(Z) = \top\\
\mathsf{pv}_1(X) = n := 4 & \mathsf{pv}_1(Y) = \forall m'{:}N.\, n, m := 4, m' & \mathsf{pv}_1(Z) = \top\\
\mathsf{pv}_2(X) = n := 4 & \mathsf{pv}_2(Y) = \forall m'{:}N.\, m := m' & \mathsf{pv}_2(Z) = \forall m'{:}N.\, n, m := 4, m'\\
\mathsf{pv}_3(X) = n := 4 & \mathsf{pv}_3(Y) = \forall m'{:}N.\, m := m' & \mathsf{pv}_3(Z) = n, m := n, m\\
\mathsf{pv}_4(X) = n := 4 & \mathsf{pv}_4(Y) = \forall m'{:}N.\, m := m' & \mathsf{pv}_4(Z) = \bot\\
\mathsf{gpv}(X) = n := 4 & \mathsf{gpv}(Y) = \forall m'{:}N.\, m := m' & \mathsf{gpv}(Z) = \bot
\end{array}$$

Parameter n is not constant in the equation for Y, since its value may change through the PVI $Y(n+7, m)$. That updated value can also reach Z through the PVI $Z(n, m)$,

hence n is also not constant in Z. Applying the global propagated values yields:

$$\begin{aligned}\mu X(n{:}N) &= \forall m'{:}N.\, Y(4, m')\\ \nu Y(n, m{:}N) &= \forall m'{:}N.\, (Z(n, m') \wedge (n \geq 2 \Rightarrow Y(n/2, m')) \wedge (n \leq 5 \Rightarrow Y(n+7, m')))\\ \mu Z(n, m{:}N) &= (n < m \wedge Z(n, m-1)) \vee (n \geq m)\end{aligned}$$

After parameter elimination, which removes parameter n of X and m of Y, and pushing quantifiers inside, we obtain

$$\begin{aligned}\mu X &= Y(4)\\ \nu Y(n{:}N) &= (\forall m'{:}N.\, Z(n, m')) \wedge (n \geq 2 \Rightarrow Y(n/2)) \wedge (n \leq 5 \Rightarrow Y(n+7))\\ \mu Z(n, m{:}N) &= (n < m \wedge Z(n, m-1)) \vee (n \geq m)\end{aligned}$$

This PBES has fewer parameters and should thus be easier to solve. In particular, we avoid the instantiation of many nodes $Y(n, m)$ for all different values of m. For $Z(n, m)$, though, we still require infinitely many nodes for all values of m. □

We continue by proving the correctness of the function global-prop. First, we show that the computation of global propagated values terminates.

Theorem 7.19. *Quantified predicate analysis terminates for every PBES $\mathcal{E}$.*

Proof. Let $\mathcal{E}$ be a PBES. In every iteration i, there is some $X \in \mathsf{bnd}(\mathcal{E})$ for which one of the following happens:

- $\mathsf{pv}_i(X)$ changes from $\top$ to $\mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e}$; or
- otherwise, if $\mathsf{pv}_i(X) = \mathbf{Q}.\, \mathbf{d} :\overset{X}{=} \mathbf{e}$, then either some $\mathbf{d}[i]$ is removed from $\mathbf{d}$ or some prefix from $\mathbf{Q}$ is removed (or both).

Since $\mathcal{E}$ consists of a finite number of equations and each equation has a finite number of parameters, this process must terminate within a finite number of iterations. □

We now continue proving that the function global-prop preserves PBES semantics. In the reasoning, we reuse some of the lemmata from Section 7.4 and introduce several new lemmata. Furthermore, we apply the *switching rule* [59, Lemma 21], which states that adjacent equations may be swapped if they have the same fixpoint symbol. In formal terms, that corresponds to the following equality for all η, δ and $\mathcal{E}$ such that $X, Y \notin \mathsf{bnd}(\mathcal{E})$:

$$[\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}]\!]\eta\delta = [\![(\sigma Y(d{:}D) = \varphi_Y)(\sigma X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta$$

The validity of the switching rule follows quickly from Bekič's lemma on simultaneous fixpoints [13] (see also the discussion preceding Lemma 7.13).

The first lemma claims that the solution of a PBES $\mathcal{E}$ indeed satisfies the equivalence specified by each equation. Similar to earlier lemmata, this lemma follows from the more general result on fixpoint equation systems [117].

Lemma 7.20. *[117, Lemma 12] Let $\mathcal{E}$ be a PBES and η and δ arbitrary environments. If $\eta_{\mathcal{E}} = [\![\mathcal{E}]\!]\eta\delta$, then for all $X \in \mathsf{bnd}(\mathcal{E})$ we have $\eta_{\mathcal{E}}(X) = \{v \in \mathbb{D} \mid [\![\varphi_X]\!]\eta_{\mathcal{E}}\delta[v/d]\}$.*

Proof. By structural induction on $\mathcal{E}$. The base case, where $\mathcal{E} = \emptyset$, vacuously holds since $\mathsf{bnd}(\emptyset) = \emptyset$. For the induction step, let $\mathcal{E} = (\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}'$ for some $\mathcal{E}'$. Assume as induction hypothesis that for all $X \in \mathsf{bnd}(\mathcal{E}')$ and environments η and δ, we have

$$([\![\mathcal{E}']\!]\eta\delta)(X) = \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}']\!]\eta\delta)\delta[v/d]\} \tag{$\dagger$}$$

Let $X \in \mathsf{bnd}(\mathcal{E})$ be arbitrary; we distinguish two cases:

- $X \neq Y$. Then it has to hold that $X \in \mathsf{bnd}(\mathcal{E}')$. Below, let η and δ be arbitrary and let $T_Y(R) = \{v \in \mathbb{D} \mid [\![\varphi_Y]\!]([\![\mathcal{E}']\!]\eta[R/Y]\delta)\delta[v/d]\}$ be the predicate transformer for Y.

$$\begin{aligned}
&([\![(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta)(X)\\
&= ([\![\mathcal{E}']\!]\eta[\sigma T_Y/Y]\delta)(X)\\
&\overset{(\dagger)}{=} \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}']\!]\eta[\sigma T_Y/Y]\delta)\delta[v/d]\}\\
&= \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}']\!]\eta\delta)\delta[v/d]\}
\end{aligned}$$

- $X = Y$. Then we have $X \notin \mathsf{bnd}(\mathcal{E}')$. Let η and δ again be arbitrary and let $T_X(R) = \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}']\!]\eta[R/X]\delta)\delta[v/d]\}$ be the predicate transformer for X. In the deduction below, $*$ indicates the application of Lemma 7.9.

$$\begin{aligned}
&([\![(\sigma X(d{:}D) = \varphi_X)\mathcal{E}']\!]\eta\delta)(X)\\
&= ([\![\mathcal{E}']\!]\eta[\sigma T_X/X]\delta)(X)\\
&\overset{(*)}{=} \sigma T_X\\
&= T_X(\sigma T_X)\\
&= \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}']\!]\eta[\sigma T_X/X]\delta)\delta[v/d]\}\\
&= \{v \in \mathbb{D} \mid [\![\varphi_X]\!]([\![(\sigma X(d{:}D) = \varphi_X)\mathcal{E}']\!]\eta\delta)\delta[v/d]\}
\end{aligned}$$

□

We slightly restate the result from the previous lemma, so it follows the format of the assumptions in Lemma 7.11.

Lemma 7.21. *Given a PBES $\mathcal{E}$ and a PVI $X(\mathbf{e})$, with $X \in \mathsf{bnd}(\mathcal{E})$, the following holds:*

$$[\![X(\mathbf{e})]\!]\eta_{\mathcal{E}}\delta = [\![\varphi_X]\!]\eta_{\mathcal{E}}\delta[[\![\mathbf{e}]\!]\delta/\mathbf{d}_X]$$

where $\eta_{\mathcal{E}} = [\![\mathcal{E}]\!]$ and δ is an arbitrary data environment.

Proof. We use Lemma 7.20 to perform the following deduction.

$$\begin{aligned}
&[\![X(\mathbf{e})]\!]\eta_{\mathcal{E}}\delta\\
&\Leftrightarrow [\![\mathbf{e}]\!]\delta \in \eta_{\mathcal{E}}(X)\\
&\Leftrightarrow [\![\mathbf{e}]\!]\delta \in \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]\eta_{\mathcal{E}}\delta[\mathbf{v}/\mathbf{d}_X]\}\\
&\Leftrightarrow [\![\varphi_X]\!]\eta_{\mathcal{E}}\delta[[\![\mathbf{e}]\!]\delta/\mathbf{d}_X]
\end{aligned}$$

□

We present another variant of the substitution rule, which allows substituting φ_X for X in the right-hand side of Y, even though Y occurs after X, as long as X and Y are adjacent and have the same fixpoint symbol. We refer to this rule as the *adjacent substitution rule*.

Lemma 7.22. *Let $(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}$ be a PBES where $X, Y \notin \mathsf{bnd}(\mathcal{E})$. Then, for all η and δ, it holds that*

$$[\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}]\!]\eta\delta = [\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y[\varphi_X/X])\mathcal{E}]\!]\eta\delta$$

Proof. Let X, Y and $\mathcal{E}$ be as above and let η and δ be arbitrary. We perform the following derivation, where $*$ indicates use of the switching rule and † indicates use of the substitution rule.

$$\begin{aligned}
&[\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y)\mathcal{E}]\!]\eta\delta \\
\overset{(*)}{=}\ &[\![(\sigma Y(d{:}D) = \varphi_Y)(\sigma X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta \\
\overset{(\dagger)}{=}\ &[\![(\sigma Y(d{:}D) = \varphi_Y[\varphi_X/X])(\sigma X(d{:}D) = \varphi_X)\mathcal{E}]\!]\eta\delta \\
\overset{(*)}{=}\ &[\![(\sigma X(d{:}D) = \varphi_X)(\sigma Y(d{:}D) = \varphi_Y[\varphi_X/X])\mathcal{E}]\!]\eta\delta
\end{aligned}$$

□

In similar fashion as Lemma 7.14, the next lemma shows when a QPVI $\mathbf{Q}.\, X(\mathbf{e})$ can be replaced by $\mathbf{Q}.\, \tilde{X}(\mathbf{e})$ in the right-hand side of $\tilde{X}$.

Lemma 7.23. *Let $\mathcal{E} = (\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma \tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p])\mathcal{E}_2$ be a PBES and $\mathbf{Q}.\, X(\mathbf{e})$ a QPVI such that $\mathbf{Q}.\, X(\mathbf{e}) \hookrightarrow \mathbf{Q}_p.\, \mathbf{d}_p := \mathbf{e}_p$. Then*

$$[\![\mathcal{E}]\!]\eta\delta = [\![(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)(\sigma \tilde{X}(\mathbf{d}{:}\mathbf{D}) = \mathbf{Q}_p.\, \varphi_X[\mathbf{e}_p/\mathbf{d}_p][\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})])\mathcal{E}_2]\!]\eta\delta$$

holds for all η and δ.

Proof. The proof follows the same reasoning as the proof of Lemma 7.14, but applies the adjacent substitution rule of Lemma 7.22, instead of the (self-)substitution rule. □

The final lemma before the main theorem provides another result on the substitution of X by $\tilde{X}$.

Lemma 7.24. *Let $\mathcal{E} = \mathcal{E}_1\mathcal{E}_2$ be a closed PBES such that $X, \tilde{X} \in \mathsf{bnd}(\mathcal{E}_2)$. If it holds that $[\![\mathbf{Q}.\, X(\mathbf{e})]\!]([\![\mathcal{E}_2]\!]\eta\delta')\delta = [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]([\![\mathcal{E}_2]\!]\eta\delta')\delta$ for all η, δ and δ', then $[\![\mathcal{E}]\!] = [\![\mathcal{E}_1[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]$.*

Proof. By induction on the length of $\mathcal{E}_1$. In the base case, where $\mathcal{E}_1 = \emptyset$, then the substitution $[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]$ may trivially be applied to $\mathcal{E}_1$. Otherwise, if $\mathcal{E}_1 = (\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)\mathcal{E}'$, we assume as induction hypothesis that $[\![\mathcal{E}'\mathcal{E}_2]\!]\eta\delta = [\![\mathcal{E}'[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta\delta$ for all η and δ.

By the PBES semantics, for all η and δ, there is some η' such that $[\![\mathcal{E}'\mathcal{E}_2]\!]\eta\delta = [\![\mathcal{E}_2]\!]\eta'\delta$ and $\eta(X) = \eta'(X)$ (and likewise for $\tilde{X}$). Hence, we obtain for all η, δ and δ' that

$$[\![\mathbf{Q}.\, X(\mathbf{e})]\!]([\![\mathcal{E}'\mathcal{E}_2]\!]\eta\delta')\delta = [\![\mathbf{Q}.\, \tilde{X}(\mathbf{e})]\!]([\![\mathcal{E}'\mathcal{E}_2]\!]\eta\delta')\delta \qquad (*)$$

Let η and δ be arbitrary environments. We have the following predicate transformers:

$$\begin{aligned}
T_X(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X]\!]([\![\mathcal{E}'\mathcal{E}_2]\!]\eta[R/X]\delta)\delta[\mathbf{v}/\mathbf{d}]\}\\
T'_X(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]([\![\mathcal{E}'\mathcal{E}_2]\!]\eta[R/X]\delta)\delta[\mathbf{v}/\mathbf{d}]\}\\
T''_X(R) &= \{\mathbf{v} \in \mathbb{D} \mid [\![\varphi_X[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]]\!]\\
&\qquad\qquad ([\![\mathcal{E}'[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta[R/X]\delta)\delta[\mathbf{v}/\mathbf{d}]\}
\end{aligned}$$

We have for all R that $T_X(R) \overset{(*)}{=} T'_X(R) \overset{(IH)}{=} T''_X(R)$. Thus it holds that $\sigma T_X = \sigma T''_X$, and we can derive

$$\begin{aligned}
&[\![\mathcal{E}_1\mathcal{E}_2]\!]\eta\delta\\
&= [\![(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X)\mathcal{E}'\mathcal{E}_2]\!]\eta\delta\\
&= [\![\mathcal{E}'\mathcal{E}_2]\!]\eta[\sigma T_X/X]\delta\\
&\overset{(IH)}{=} [\![\mathcal{E}'[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta[\sigma T_X/X]\delta\\
&= [\![\mathcal{E}'[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta[\sigma T''_X/X]\delta\\
&= [\![(\sigma X(\mathbf{d}{:}\mathbf{D}) = \varphi_X[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})])\mathcal{E}'[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta\delta\\
&= [\![\mathcal{E}_1[\mathbf{Q}.\, \tilde{X}(\mathbf{e})/\mathbf{Q}.\, X(\mathbf{e})]\mathcal{E}_2]\!]\eta\delta
\end{aligned}$$

□

Now we are ready to show that the gpv function preserves the semantics of the PBES it transforms. Below, we assume the PBES $\mathcal{E}$ is such that guard returns a satisfiable formula for any $X(\mathbf{e}')$ and $\mathbf{Q}.\, \varphi_Y[\mathbf{e}/\mathbf{d}]$, where $\mathsf{gpv}(X) = \mathbf{Q}.\, \mathbf{d} := \mathbf{e}$ and $X(\mathbf{e}) \in \mathsf{qiocc}(\mathbf{Q}.\, \varphi_Y[\mathbf{e}/\mathbf{d}])$. The proof can be extended to include the case where guard may return an unsatisfiable formula; in that case the PVI $X(\mathbf{e})$ may be replaced by any other formula in $\mathbf{Q}.\, \varphi_Y[\mathbf{e}/\mathbf{d}]$ (see Section 7.6).

Theorem 7.25. *Let $\mathcal{E}$ be a closed PBES and $\hat{X}(\hat{\mathbf{e}})$ some target node. Then $[\![\hat{\mathbf{e}}]\!] \in [\![\mathcal{E}]\!](\hat{X})$ if and only if $[\![\hat{\mathbf{e}}]\!] \in [\![\mathsf{global\text{-}prop}(\mathcal{E})]\!](\hat{X})$.*

Proof. As above, let $\mathcal{E}$ be a PBES and $\hat{X}(\hat{\mathbf{e}})$ a target node. Some predicate variable $X \in \mathsf{bnd}(\mathcal{E})$ is irrelevant for the solution of $\hat{X}(\hat{\mathbf{e}})$ if there is no sequence of predicate variables $X_1 \dots X_n$ such that $\hat{X} = X_1$, $X = X_n$ and X_i occurs in the right-hand side $\varphi_{X_{i-1}}$ for all $1 < i \leq n$. Such a sequence does not exist if and only if $\mathsf{gpv}(X) = \top$. Hence, the equation of any predicate variable X for which $\mathsf{gpv}(X) = \top$ can be eliminated from $\mathcal{E}$, according to the definition of global-prop, without affecting the solution of $\hat{X}(\hat{\mathbf{e}})$.

Below, we assume that $\mathcal{E}$ is defined as $(\sigma_1 X_1(\mathbf{d}_1{:}\mathbf{D}_1) = \varphi_1) \dots (\sigma_n X_n(\mathbf{d}_n{:}\mathbf{D}_n) = \varphi_n)$ and that we have $\mathsf{gpv}(X_i) = \mathbf{Q}_i.\, \mathbf{d}_i := \mathbf{e}_i$ for all $1 \leq i \leq n$. Let k be the index of $\hat{X}$ in $\mathcal{E}$, *i.e.*, $\hat{X} = X_k$.

Then, we define $\mathcal{E}'$ to be the PBES in which all equation are duplicated, the global propagated values are applied to the new equations and the quantifiers are pushed inside; it is formally defined as:

$$\begin{aligned}\mathcal{E}' = \big(\sigma_1 X_1(\mathbf{d}_1{:}\mathbf{D}_1) = \varphi_1\big)\big(\sigma_1 \tilde{X}_1(\mathbf{d}_1{:}\mathbf{D}_1) = \tilde{\varphi}_1\big)\\ \vdots\\ \big(\sigma_n X_n(\mathbf{d}_n{:}\mathbf{D}_n) = \varphi_n\big)\big(\sigma_n \tilde{X}_n(\mathbf{d}_n{:}\mathbf{D}_n) = \tilde{\varphi}_n\big)\end{aligned}$$

where, for each $1 \leq i \leq n$, $\tilde{\varphi}_i = \mathsf{qi}(\mathbf{Q}_i.\, \varphi_i[\mathbf{e}_i/\mathbf{d}_i])$. Since we only added equations to $\mathcal{E}$ to obtain $\mathcal{E}'$, it holds that $[\![\hat{\mathbf{e}}]\!] \in [\![\mathcal{E}]\!](\hat{X})$ if and only if $[\![\hat{\mathbf{e}}]\!] \in [\![\mathcal{E}']\!](\hat{X})$

By definition of gpv and $\wedge$, we have $\hat{X}(\hat{\mathbf{e}}) \hookrightarrow \mathsf{gpv}(\hat{X})$. Thus, if $\eta = [\![\mathcal{E}']\!]$, can derive the following (where $*$ indicates the application of Lemma 7.21):

$$\begin{aligned}&[\![\hat{X}(\hat{\mathbf{e}})]\!]\eta\delta\\ \overset{(*)}{=}\ &[\![\varphi_k]\!]\eta\delta[[\![\hat{\mathbf{e}}]\!]\delta/\mathbf{d}_X]\\ =\ &[\![\varphi_k[\mathbf{e}_k/\mathbf{d}_k]]\!]\eta\delta[[\![\hat{\mathbf{e}}]\!]\delta/\mathbf{d}_X]\\ =\ &[\![\mathsf{qi}(\varphi_k[\mathbf{e}_k/\mathbf{d}_k])]\!]\eta\delta[[\![\hat{\mathbf{e}}]\!]\delta/\mathbf{d}_X]\\ =\ &[\![\tilde{\varphi}_k]\!]\eta\delta[[\![\hat{\mathbf{e}}]\!]\delta/\mathbf{d}_X]\\ \overset{(*)}{=}\ &[\![\tilde{X}_k(\hat{\mathbf{e}})]\!]\eta\delta\end{aligned}$$

We consider some predicate variable $X_i \in \mathsf{bnd}(\mathcal{E})$ and show that all its occurrences in the new right-hand sides $\tilde{\varphi}_j$, with $1 \leq j \leq n$, can be replaced by the corresponding variable $\tilde{X}_i$, while preserving the semantics of $\mathcal{E}'$. Let $\mathbf{Q}.\, X_i(\mathbf{e})$ be a QPVI that occurs in $\mathcal{E}'$, such that $\mathbf{Q}.\, X_i(\mathbf{e}) \hookrightarrow \mathbf{Q}_i.\, \mathbf{d}_i := \mathbf{e}_i$. Furthermore, let $\mathcal{E}_i$ be the sequence of equations that occur before and $\mathcal{E}'_i$ the sequence of equations that occur after $\tilde{X}_i$ in $\mathcal{E}'$. By the subsequent application of Lemmas 7.13 and 7.23, we have for all η and δ:

$$\begin{aligned}&[\![(\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i)\mathcal{E}'_i]\!]\eta\delta\\ =\ &[\![(\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i)\mathcal{E}'_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})]]\!]\eta\delta\\ =\ &[\![(\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})]\\ &\qquad\qquad \mathcal{E}'_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})]]\!]\eta\delta\end{aligned}$$

By Lemma 7.21, the environment $\eta' = [\![(\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i)\mathcal{E}'_i]\!]\eta\delta$ satisfies $[\![X_i(\mathbf{e})]\!]\eta'\delta' = [\![\varphi_i]\!]\eta'\delta'[[\![\mathbf{e}]\!]\delta'/\mathbf{d}_i]$ for all η, δ and δ'. The same holds for $\tilde{X}_i$. Thus, with Lemmas 7.11 and 7.24, we can derive that

$$\begin{aligned}&[\![\mathcal{E}_i(\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i)\mathcal{E}'_i]\!]\eta\delta\\ =\ &[\![\mathcal{E}_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})](\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i)\mathcal{E}'_i]\!]\eta\delta\\ =\ &[\![\mathcal{E}_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})](\sigma_i X_i(\mathbf{d}_i{:}\mathbf{D}_i) = \varphi_i)(\sigma_i \tilde{X}_i(\mathbf{d}_i{:}\mathbf{D}_i) = \tilde{\varphi}_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})])\\ &\qquad\qquad \mathcal{E}'_i[\mathbf{Q}.\, \tilde{X}_i(\mathbf{e})/\mathbf{Q}.\, X_i(\mathbf{e})]]\!]\eta\delta\end{aligned}$$

The definitions of gpv and $\wedge$ imply that, for all QPVIs $\mathbf{Q}'.\, X_j(\mathbf{e}')$ that occur in a right-hand side $\tilde{\varphi}_l$, with $1 \leq l \leq n$, we have $\mathbf{Q}'.\, X_j(\mathbf{e}') \hookrightarrow \mathbf{Q}_j.\, \mathbf{d}_j := \mathbf{e}_j$. Furthermore, the above procedure can be repeated independently for each QPVI $\mathbf{Q}'.\, X_j(\mathbf{e}')$ such that $\mathbf{Q}'.\, X_j(\mathbf{e}') \hookrightarrow \mathbf{Q}_j.\, \mathbf{d}_j := \mathbf{e}_j$, since none of the involved substitutions affect the semantics of any suffix of $\mathcal{E}'$. As a result, we obtain the following PBES that has the same semantics as $\mathcal{E}'$:

$$\begin{gathered}\mathcal{E}'' = \big(\sigma_1 X_1(\mathbf{d}_1{:}\mathbf{D}_1) = \varphi_1\big)\big(\sigma_1 \tilde{X}_1(\mathbf{d}_1{:}\mathbf{D}_1) = \tilde{\varphi}_1[\tilde{X}_i/X_i]_{1\leq i\leq n}\big) \\ \vdots \\ \big(\sigma_n X_n(\mathbf{d}_n{:}\mathbf{D}_n) = \varphi_n\big)\big(\sigma_n \tilde{X}_n(\mathbf{d}_n{:}\mathbf{D}_n) = \tilde{\varphi}_n[\tilde{X}_i/X_i]_{1\leq i\leq n}\big)\end{gathered}$$

Since the predicate variables $X_1, \ldots, X_n$ do not occur in the equations for $\tilde{X}_1, \ldots, \tilde{X}_n$ and we are only interested in $[\![\hat{\mathbf{e}}]\!] \in [\![\mathcal{E}'']\!](\hat{X})$, we can eliminate the equations for $X_1, \ldots, X_n$, yielding

$$\big(\sigma_1 \tilde{X}_1(\mathbf{d}_1{:}\mathbf{D}_1) = \tilde{\varphi}_1[\tilde{X}_i/X_i]_{1\leq i\leq n}\big) \cdots \big(\sigma_n \tilde{X}_n(\mathbf{d}_n{:}\mathbf{D}_n) = \tilde{\varphi}_n[\tilde{X}_i/X_i]_{1\leq i\leq n}\big)$$

This PBES is isomorph to the following PBES, where every variable $\tilde{X}_i$ is renamed to X_i (recall that $\tilde{\varphi}_i = \mathsf{qi}(\mathbf{Q}_i.\, \varphi_i[\mathbf{e}_i/\mathbf{d}_i])$):

$$\big(\sigma_1 X_1(\mathbf{d}_1{:}\mathbf{D}_1) = \mathsf{qi}(\mathbf{Q}_i.\, \varphi_1[\mathbf{e}_1/\mathbf{d}_1])\big) \cdots \big(\sigma_n X_n(\mathbf{d}_n{:}\mathbf{D}_n) = \mathsf{qi}(\mathbf{Q}_i.\, \varphi_n[\mathbf{e}_n/\mathbf{d}_n])\big)$$

Since distribution of quantifiers over other operators, as defined in the quantifier-inside rewriter, preserves the semantics of predicate formulae, the PBES above has the same semantics as $\mathsf{global\text{-}prop}(\mathcal{E})$. Thus we have $[\![\hat{\mathbf{e}}]\!] \in [\![\mathsf{global\text{-}prop}(\mathcal{E})]\!](\hat{X})$. □

7.6 Guards for Predicate Formulae

The analysis of the previous section relies solely on static dependencies between predicate variables, by extracting all QPVIs $\mathbf{Q}.\, X(\mathbf{e})$ in a right-hand side with the function qiocc. It does not take into account any other parts of the predicate formula, which might cause subformula $X(\mathbf{e})$ to be irrelevant. For example, in the formula $n \leq 2 \wedge X(m)$, $X(m)$ is irrelevant if the constant $n = 7$ was deduced. A simple formula that characterises for which values of m the PVI $X(m)$ is relevant, is called a *guard* (a formal definition follows).

Guards also play an important role in other types of static analysis of PBESs. The concept of guards first appeared in [106], although it only shows how to construct a guard for a normal form called *predicate formula normal form* (PFNF). Keiren *et al.* [74] propose a function that over-approximates guards for arbitrary predicate formulae. Correctness results for this guard function can be found in Keiren's thesis [73]. Here, we improve on these results by strengthening the guards we compute and providing a detailed proof of their compositionality.

Before we give a formal definition of a guard, we first introduce several auxiliary notions. Firstly, $\rightarrow$ denotes *logical implication* on predicate formulae: $\varphi \rightarrow \psi$ iff for

all η and δ, $[\![\varphi]\!]\eta\delta \Rightarrow [\![\psi]\!]\eta\delta$. Similarly, we have *logical equivalence*: $\varphi \equiv \psi$ iff for all η and δ, $[\![\varphi]\!]\eta\delta = [\![\psi]\!]\eta\delta$. Given a predicate formula φ, an occurrence $Y(e)$ in φ is called a *predicate variable instance* (PVI). Furthermore, the number of PVIs occurring in φ is denoted with $\mathsf{npred}(\varphi)$ and the ith PVI in φ is $\mathsf{PVI}(\varphi, i)$. To replace the ith PVI in φ with an arbitrary formula ψ, we define the following syntactic replacement function.

Definition 7.26. Given a formula φ and $1 \leq i \leq \mathsf{npred}(\varphi)$, the syntactic replacement of the ith PVI in φ by ψ is denoted $\varphi[i \mapsto \psi]$, which is defined inductively as follows:

$$
\begin{aligned}
X(e)[1 \mapsto \psi] &= \psi \\
(\neg\varphi)[i \mapsto \psi] &= \neg\varphi[i \mapsto \psi] \\
(\varphi_1 \wedge \varphi_2)[i \mapsto \psi] &= \begin{cases} \varphi_1[i \mapsto \psi] \wedge \varphi_2 & \text{if } i \leq \mathsf{npred}(\varphi_1) \\ \varphi_1 \wedge \varphi_2[i - \mathsf{npred}(\varphi_1) \mapsto \psi] & \text{if } i > \mathsf{npred}(\varphi_1) \end{cases} \\
(\varphi_1 \vee \varphi_2)[i \mapsto \psi] &= \begin{cases} \varphi_1[i \mapsto \psi] \vee \varphi_2 & \text{if } i \leq \mathsf{npred}(\varphi_1) \\ \varphi_1 \vee \varphi_2[i - \mathsf{npred}(\varphi_1) \mapsto \psi] & \text{if } i > \mathsf{npred}(\varphi_1) \end{cases} \\
(\forall d{:}D.\, \varphi)[i \mapsto \psi] &= \forall d{:}D.\, \varphi[i \mapsto \psi] \\
(\exists d{:}D.\, \varphi)[i \mapsto \psi] &= \exists d{:}D.\, \varphi[i \mapsto \psi]
\end{aligned}
$$

Remark that, in the definition above, we do not need to define the case $b[i \mapsto \psi]$, since $i \leq \mathsf{npred}(\varphi)$ ensures the recursion cannot arrive at a term b. Given an expression $\varphi[i \mapsto \psi]$, we call φ the *body* of the substitution. To denote the simultaneous substitution of the ith PVI by some formula ψ_i for all $1 \leq i \leq \mathsf{npred}(\varphi)$ that satisfy the condition f, we write $\varphi[i \mapsto \psi_i]_{f(i)}$.

We are now ready to formally introduce *guards* and *free guards*. The definition of the former originates from [73, Lemma 6.27].

Definition 7.27. Given a predicate formula φ, a simple formula ψ is a *guard* for the ith PVI of φ if and only if $\varphi \equiv \varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]$.

Definition 7.28. Given a predicate formula φ, a simple formula ψ is a *free guard* for the ith PVI of φ if and only if for all environments η and δ, it holds

$$\neg[\![\psi]\!]\delta \Rightarrow [\![\varphi[i \mapsto \mathit{false}]]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{true}]\!]\eta\delta$$

To understand the intuition behind (free) guards, consider an equation $\sigma X(d{:}D) = \varphi$, the ith PVI in φ, $\mathsf{PVI}(\varphi, i) = Y(e)$ and associated guard ψ and free guard ψ_f. The guard ψ expresses for which values of e the PVI $Y(e)$ is relevant in φ. After all, if $[\![\psi]\!]\delta$ is *false*, then $[\![Y(e)]\!]\eta\delta$ is not relevant to the value of the subformula $\psi \wedge Y(e)$.

For our free guard ψ_f, we have that whenever $[\![\psi_f]\!]\delta$ becomes *false* for some δ, then the subformula $Y(e)$ is not relevant for the evaluation of φ. Note, however, that $[\![Y(e)]\!]\eta\delta$ can still influence $[\![\varphi]\!]\eta\delta$ if $Y(e)$ also occurs elsewhere in φ. On the other hand, if $[\![\psi_f]\!]\delta$ holds, we cannot conclude anything. A meaningful free guard typically only contains variables that occur freely in φ.

Thus, in general, a (free) guard over-approximates the true dependency between X and Y, as far as that dependency originates from the occurrence $\mathsf{PVI}(\varphi, i)$. Remark that *true* is a (free) guard for any PVI. We further demonstrate the concepts in the next examples.

Example 7.29. Consider the predicate formula

$$\varphi = (\exists m{:}N.\, m = 6 \wedge (W(m) \vee X)) \vee (Y \wedge (\neg b \Rightarrow Z))$$

The PVIs $W(m)$ and X are guarded in the same way: $m = 6$ is a guard for both these PVIs. From this, we can deduce that replacing $W(m)$ by $W(6)$ preserves logical equivalence of φ. Observe, however, that the only occurrence of m is not free, thus $m = 6$ is not a free guard for these PVIs; their only free guard is *true*. For Y, the only (free) guard is *true*; Z also has $\neg b$ both as guard and free guard. Hence, if we somehow deduce that b never takes on the value *false*, the PVI Z can be eliminated from φ. □

The following example shows that the substitution applied in the definition of a guard can yield unexpected results when a variable d has both bound and free occurrences.

Example 7.30. Consider the formula $\varphi = (n < 2) \wedge \forall n{:}N.\, X(n)$. The PVI $X(n)$ occurs in the conjunctive context of $n < 2$, which at first sight leads to believe that $n < 2$ is a guard. However, we have the logical inequivalence $\varphi \not\equiv \varphi[1 \mapsto (n < 2) \wedge X(n)]$, since the new occurrence of n introduced by the substitution is bound instead of free. Hence, $n < 2$ is not a guard; it is a free guard, though. □

To identify situations such as in the latter example, we say φ is *capture-avoiding* iff no free variable of φ has a bound occurrence in φ, *i.e.*, there is no subformula $\mathsf{Q}d{:}D.\, \varphi'$ of φ such that $d \in \mathsf{vars}(\varphi)$. Remark that any predicate formula can be transformed into an equivalent capture-avoiding formula by performing the appropriate alpha conversion, that is, renaming the variables bound in quantifiers.

Both examples together demonstrate that the concepts of guards and free guards are in general incomparable. The following proposition gives necessary conditions under which the notions can be related.

Proposition 7.31. *Let φ be a predicate formula, ψ a simple formula and $X(e) = \mathsf{PVI}(\varphi, i)$ the ith PVI in φ, which we assume occurs monotonically. If φ is capture-avoiding, $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$ and ψ is a free guard for the ith PVI in φ, then ψ is also a guard for the ith PVI in φ.*

Proof. Let φ, ψ and $X(e) = \mathsf{PVI}(\varphi, i)$ be as above and let η and δ be arbitrary environments. We start by stating that ψ is a free guard:

$$\neg[\![\psi]\!]\delta \Rightarrow [\![\varphi[i \mapsto \mathit{true}]]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{false}]]\!]\eta\delta$$

By the monotonic occurrence of $\mathsf{PVI}(\varphi, i)$ in φ, this implies:

$$\neg[\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{false}]]\!]\eta\delta$$

We add the trivially true statement that $[\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{true} \wedge X(e)]]\!]\eta\delta$:

$$\begin{aligned}\neg[\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta &= [\![\varphi[i \mapsto \mathit{false} \wedge X(e)]]\!]\eta\delta\\ \wedge\ [\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta &= [\![\varphi[i \mapsto \mathit{true} \wedge X(e)]]\!]\eta\delta\end{aligned}$$

Since $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$ and the fact that φ is capture-avoiding implies that its free variables are not bound, we can substitute ψ for *false* and *true* respectively:

$$\begin{aligned}\neg[\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta &= [\![\varphi[i \mapsto \psi \wedge X(e)]]\!]\eta\delta\\ \wedge\ [\![\psi]\!]\delta \Rightarrow [\![\varphi]\!]\eta\delta &= [\![\varphi[i \mapsto \psi \wedge X(e)]]\!]\eta\delta\end{aligned}$$

By resolution, this is equivalent to:

$$[\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \psi \wedge X(e)]]\!]\eta\delta$$

We conclude that ψ is also a guard. □

We remark that the reasoning in the above proof can also be used to show that, under the same conditions, if ψ satisfies $\varphi[i \mapsto \psi \Rightarrow X(e)] \equiv \varphi[i \mapsto \psi \wedge X(e)]$ (which is a stronger notion of guard), then ψ is also a free guard. In the remainder of this section, we will primarily focus on guards.

7.6.1 Compositionality

Before we show how to compute a guard, we first discuss the topic of compositionality. Given a predicate formula φ and guards $\psi_1, \ldots, \psi_n$ for each of the PVIs in φ, then $\psi_1, \ldots, \psi_n$ are *compositional* iff $\varphi \equiv \varphi[i \mapsto \psi_i \wedge \mathsf{PVI}(\varphi, i)]_{i \leq \mathsf{npred}(\varphi)}$. When computing guards, it is desirable that they are compositional. Otherwise, one needs to compute a guard for the first PVI in φ, construct $\varphi[1 \mapsto \psi_i \wedge \mathsf{PVI}(\varphi, 1)]$, compute a guard for the second PVI in $\varphi[1 \mapsto \psi_i \wedge \mathsf{PVI}(\varphi, 1)]$, apply it, and so forth for all the PVIs. This is not a scalable approach for PBESs with large right-hand sides. Our definition of guards (Definition 7.27) does not necessarily result in compositional guards, as demonstrated by the below example.

Example 7.32. Consider the formula $\varphi = X \vee X$. According to Definition 7.27, the simple formula *false* is a guard for either PVI, since $X \vee X \equiv (X \vee X)[1 \mapsto \mathit{false} \wedge X] \equiv (X \vee X)[2 \mapsto \mathit{false} \wedge X]$. However, these guards cannot both be applied to φ at the same time, since $X \vee X \not\equiv (X \vee X)[1 \mapsto \mathit{false} \wedge X][2 \mapsto \mathit{false} \wedge X] \equiv \mathit{false}$. Hence, they are not compositional. □

In the example, the fact that the formula φ contains two PVIs for the same predicate variable X is crucial. After all, if X occurs just once in φ, and the associated guard is false under δ, then the value of $\eta(X)$ does not influence $[\![\varphi]\!]\eta\delta$. Thus, we have the following proposition.

Proposition 7.33. *Let φ be a monotone predicate formula, $n = \mathsf{npred}(\varphi)$ and $\psi_1, \ldots, \psi_n$ guards for each of the respective PVIs in φ. If for every predicate variable $X \in \mathcal{X}$, at most one PVI of X occurs in φ, then $\psi_1, \ldots, \psi_n$ are compositional.*

Proof. Let φ and $\psi_1, \ldots, \psi_n$ be as above. Here, we denote each PVI $\mathsf{PVI}(\varphi, i)$ as $X_i(e_i)$. The uniqueness requirement implies that for all $i \neq j$, we have $X_i \neq X_j$.

Let $\varphi_k = \varphi[i \mapsto \psi_i \wedge X_i(e_i)]_{i \leq k}$ for all $k \leq n$. We perform induction on k. In case $k = 0$, we have the syntactic equivalence $\varphi = \varphi_0$ and immediately conclude $\varphi \equiv \varphi_0$. For the induction step, we assume that $\varphi \equiv \varphi_{k-1}$ for some $k \leq n$. In the deduction below, let η and δ be arbitrary environments. Remark that we have the syntactic equivalence $\phi_k = \phi[i \mapsto \psi_i \wedge X_i(e_i)]_{i \leq k-1}[k \mapsto \psi_k \wedge X_k(e_k)]$, since the two substitutions do not interfere with each other. Thus we derive:

$$\begin{aligned} & [\![\varphi_k]\!]\eta\delta \\ \Leftrightarrow\ & [\![\varphi_{k-1}[k \mapsto \psi_k \wedge X_k(e_k)]]\!]\eta\delta \end{aligned}$$

Assume that $X_k(e_k)$ occurs in the scope of the quantifiers $\mathsf{Q}_1 d_1{:}D_1 \ldots \mathsf{Q}_n d_n{:}D_n$. Since X_k only occurs once in φ_{k-1}, the substitution $[k \mapsto \psi_k \wedge X_k(e_k)]$ can also be captured in a predicate environment η', where

$$\eta'(X_k) = \eta(X_k) \cap \bigcup_{v \in \mathbb{D}_1 \times \cdots \times \mathbb{D}_n} \{[\![e_k]\!]\delta[v/d_1, \ldots, d_n] \mid [\![\psi_k]\!]\delta[v/d_1, \ldots, d_n]\}$$

and $\eta'(X_j) = \eta(X_j)$ for all $j \neq k$. We continue the derivation and use η':

$$\begin{aligned} & [\![\varphi_{k-1}[k \mapsto \psi_k \wedge X_k(e_k)]]\!]\eta\delta \\ \Leftrightarrow\ & [\![\varphi_{k-1}]\!]\eta'\delta \\ \overset{(IH)}{\Leftrightarrow}\ & [\![\varphi]\!]\eta'\delta \end{aligned}$$

The PVI $X_k(e_k)$ still occurs in the scope of the same quantifiers, so we can reintroduce the substitution $[k \mapsto \psi_k \wedge X_k(e_k)]$.

$$\begin{aligned} & [\![\varphi]\!]\eta'\delta \\ \Leftrightarrow\ & [\![\varphi[k \mapsto \psi_k \wedge X_k(e_k)]]\!]\eta\delta \\ \Leftrightarrow\ & [\![\varphi]\!]\eta\delta \end{aligned}$$

This completes the induction step and, thereby, the proof. □

7.6.2 Computing Guards

For formulas in PFNF, a guard can be obtained trivially [106]. Let

$$\mathsf{Q}_1 d_1{:}D_1 \ldots \mathsf{Q}_n d_n{:}D_n.\, h \wedge \bigwedge_{i \in I} \Big(g_i \Rightarrow \bigvee_{j \in J_i} X^j(\mathbf{e}^j) \Big)$$

be a formula in PFNF. Then a guard for X^j, where $j \in J_i$, is $h \wedge g_i$. The same applies to disjunctive and conjunctive predicate formulae, as used in the SRF normal form. Computing a meaningful guard for a predicate formula with arbitrary structure is slightly more involved. For this we propose the function $\mathsf{guard}^i(\varphi)$.

Definition 7.34. Let φ be a normalised, capture-avoiding predicate formula and $i \leq \mathsf{npred}(\varphi)$. Then, the function guard is defined inductively as follows:

$$\begin{aligned}
\mathsf{guard}^i(Y(\mathbf{e})) &= \mathit{true} \\
\mathsf{guard}^i(\varphi \wedge \psi) &= \begin{cases} s(\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \text{if } i > \mathsf{npred}(\varphi) \\ s(\psi) \wedge \mathsf{guard}^i(\varphi) & \text{if } i \leq \mathsf{npred}(\varphi) \end{cases} \\
\mathsf{guard}^i(\varphi \vee \psi) &= \begin{cases} s(\neg\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \text{if } i > \mathsf{npred}(\varphi) \\ s(\neg\psi) \wedge \mathsf{guard}^i(\varphi) & \text{if } i \leq \mathsf{npred}(\varphi) \end{cases} \\
\mathsf{guard}^i(\forall d{:}D.\, \varphi) &= s(\forall d{:}D.\, \varphi) \wedge \mathsf{guard}^i(\varphi) \\
\mathsf{guard}^i(\exists d{:}D.\, \varphi) &= s(\neg\exists d{:}D.\, \varphi) \wedge \mathsf{guard}^i(\varphi)
\end{aligned}$$

where

$$\begin{aligned}
s(\varphi) &= \varphi[i \mapsto \mathit{true}]_{i \leq \mathsf{npred}(\varphi)} \\
s(\neg\varphi) &= \neg\varphi[i \mapsto \mathit{false}]_{i \leq \mathsf{npred}(\varphi)}
\end{aligned}$$

Similar to Definition 7.26, the case $\mathsf{guard}^i(b)$ cannot occur due to $i \leq \mathsf{npred}(\varphi)$. We also do not define $\mathsf{guard}^i(\neg\varphi)$, since normalised formulae do not contain negations and implications. In [74], the function s was defined as $s(\varphi) = \varphi$ if $\mathsf{npred}(\varphi) = 0$, and *true* otherwise. Furthermore, quantifiers were not taken into account. That results in guard yielding a weaker formula than our guard function does. Remark that the expressions produced by the function guard are not necessarily free guards. Take, for example, $\mathsf{guard}^1(\exists n{:}N.\, n \geq 3 \wedge X(n)) = n \geq 3$, which is not a free guard.

Before we prove that our function guard indeed yields proper, compositional guards, we first introduce several auxiliary lemmata.

Lemma 7.35. *For all monotone φ, it holds that $\varphi \to s(\varphi)$ and, dually $\neg\varphi \to s(\neg\varphi)$.*

Proof. We consider the case of $s(\varphi)$, the other case is analogous. Let φ be an arbitrary monotone formula and let η and δ be arbitrary. Furthermore, let η_{true} be such that $v \in \eta_{\mathit{true}}(X)$ for all $X \in \mathsf{occ}(\varphi)$ and $v \in \mathbb{D}$. Due to monotonicity of φ, $[\![\varphi]\!]\eta\delta \Rightarrow [\![\varphi]\!]\eta_{\mathit{true}}\delta$. Moreover, $[\![\varphi]\!]\eta_{\mathit{true}}\delta = [\![s(\varphi)]\!]\eta\delta$. Using this, we can deduce that $s(\varphi)$ is weaker than φ: $[\![\varphi]\!]\eta\delta \Rightarrow [\![\varphi]\!]\eta_{\mathit{true}}\delta = [\![s(\varphi)]\!]\eta\delta$. □

Lemma 7.36. *Let φ be a capture-avoiding predicate formula and let $I \subseteq \{1, \ldots, \mathsf{npred}(\varphi)\}$ be a set of PVI indices. Assume that we have $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$ and $\varphi \to \psi$ for some simple ψ and, for all $i \in I$, $\mathsf{PVI}(\varphi, i)$ occurs monotonically in φ. Then, it holds that $\varphi \equiv \varphi[i \mapsto (\psi \wedge \mathsf{PVI}(\varphi, i))]_{i \in I}$.*

Proof. Let φ and I and ψ be as above and let η and δ be arbitrary environments. Our proof obligation is $[\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta$. We distinguish two cases:

- If $[\![\varphi]\!]\eta\delta = \mathit{false}$, we use the monotonic occurrence of every $\mathsf{PVI}(\varphi, i)$ to derive $[\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta \Leftarrow [\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta$. It follows that $[\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta = \mathit{false}$.

- If $[\![\varphi]\!]\eta\delta = \mathit{true}$, then we use the assumption that $\varphi \to \psi$ to derive that $[\![\psi]\!]\eta\delta = \mathit{true}$. Since $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$ and φ is capture-avoiding, we can derive that $[\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{true} \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta = [\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta$. □

Lemma 7.37. *Let φ and φ' be monotone predicate formulae such that $\varphi' \to \varphi$ and let $I \subseteq \{1, \ldots, \mathsf{npred}(\varphi)\}$ be a set of PVI indices. If φ is capture-avoiding and $\mathsf{vars}(s(\neg\varphi')) \subseteq \mathsf{vars}(\varphi)$, then it holds that $\varphi \equiv \varphi[i \mapsto s(\neg\varphi') \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}$.*

Proof. Let φ, φ' and I be as above and let η and δ be arbitrary. In case $[\![\varphi]\!]\eta\delta = \mathit{false}$, we follow the same reasoning as in the proof of Lemma 7.36 to deduce that $[\![\varphi[i \mapsto s(\neg\varphi') \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta = \mathit{false}$. Henceforth, assume that $[\![\varphi]\!]\eta\delta = \mathit{true}$. We distinguish two cases:

- $[\![\varphi[i \mapsto \mathit{false}]_{i\in I}]\!]\eta\delta = [\![\varphi[i \mapsto \mathit{true}]_{i\in I}]\!]\eta\delta$, that is, none of PVIs characterised by I is relevant for the truth value of φ. From monotonicity of φ, it follows that $[\![\varphi[i \mapsto \psi]_{i\in I}]\!]\eta\delta = \mathit{true}$ for any ψ, and we conclude that $[\![\varphi[i \mapsto s(\neg\varphi) \wedge \mathsf{PVI}(\varphi, i)]]\!]\eta\delta = \mathit{true}$.

- $[\![\varphi[i \mapsto \mathit{false}]_{i\in I}]\!]\eta\delta \neq [\![\varphi[i \mapsto \mathit{true}]_{i\in I}]\!]\eta\delta$, so at least some $\mathsf{PVI}(\varphi, i)$, with $i \in I$, is relevant in φ. With monotonicity of φ and the assumption that $[\![\varphi]\!]\eta\delta = \mathit{true}$, we obtain $[\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta = [\![\psi]\!]\eta\delta$ for all ψ such that $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$.

  For the predicate formula $\varphi[j \mapsto \psi \wedge \mathsf{PVI}(\varphi, j)]_{j\leq \mathsf{npred}(\varphi)}$, we can distinguish two similar cases. The first case, where $[\![\varphi[j \mapsto \psi \wedge \mathsf{PVI}(\varphi, j)]_{j\leq \mathsf{npred}(\varphi)}]\!]\eta\delta = \mathit{true}$ for all ψ, is in contradiction with the fact that $[\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta = [\![\psi]\!]\eta\delta$ for all ψ, since monotonicity of φ yields $[\![\varphi[j \mapsto \psi \wedge \mathsf{PVI}(\varphi, j)]_{j\leq \mathsf{npred}(\varphi)}]\!]\eta\delta \Rightarrow [\![\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]_{i\in I}]\!]\eta\delta$. Hence, we conclude that only the second case can occur, so we have for all ψ such that $\mathsf{vars}(\psi) \subseteq \mathsf{vars}(\varphi)$,

$$[\![\varphi[j \mapsto \psi \wedge \mathsf{PVI}(\varphi, j)]_{j\leq \mathsf{npred}(\varphi)}]\!]\eta\delta = [\![\psi]\!]\eta\delta \qquad (\dagger)$$

  From $\varphi' \to \varphi$, we can derive that

$$[\![\varphi'[j \mapsto \mathit{false}]_{j\leq \mathsf{npred}(\varphi')}]\!]\eta\delta \Rightarrow [\![\varphi[j \mapsto \mathit{false}]_{j\leq \mathsf{npred}(\varphi)}]\!]\eta\delta \qquad (\ddagger)$$

Thus, we can derive that

$$\begin{aligned}
&[\![\varphi[i \mapsto s(\neg\varphi') \wedge \mathsf{PVI}(\varphi, i)]_{i \in I}]\!]\eta\delta \\
&= [\![\varphi[i \mapsto \neg\varphi'[j \mapsto \mathit{false}]_{j \leq \mathsf{npred}(\varphi')} \wedge \mathsf{PVI}(\varphi, i)]_{i \in I}]\!]\eta\delta \\
&\overset{(\dagger)}{=} [\![\neg\varphi'[j \mapsto \mathit{false}]_{j \leq \mathsf{npred}(\varphi')}]\!]\eta\delta \\
&= \neg[\![\varphi'[j \mapsto \mathit{false}]_{j \leq \mathsf{npred}(\varphi')}]\!]\eta\delta \\
&\overset{(\ddagger)}{\Leftarrow} \neg[\![\varphi[j \mapsto \mathit{false}]_{j \leq \mathsf{npred}(\varphi)}]\!]\eta\delta \\
&= \neg[\![\varphi[j \mapsto \mathit{false} \wedge \mathsf{PVI}(\varphi, j)]_{j \leq \mathsf{npred}(\varphi)}]\!]\eta\delta \\
&\overset{(\dagger)}{=} \neg[\![\mathit{false}]\!]\eta\delta \\
&= \mathit{true} \\
&= [\![\varphi]\!]\eta\delta
\end{aligned}$$

□

Before we continue, we make a few remarks about substitutions. In general, a substitution may change the number of PVIs in a formula or may change the PVIs themselves. However, the number of PVIs and their identities are preserved by a substitution of the shape $\varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]$ if ψ is simple. This property is used in the proof below to conclude that some substitutions may be broken up into multiple substitutions, or that their order may be swapped.

Furthermore, remark that substitution of PVIs is not a congruence under logical equivalence, as witnessed by $X \vee X \equiv X$ and $(X \vee X)[1 \mapsto \mathit{false}] \not\equiv X[1 \mapsto \mathit{false}]$. Hence, in the proof below, we can only manipulate the body of a substitution under syntactical equivalence. The next theorem shows that our function guard indeed yields compositional guards.

Theorem 7.38. *For all normalised, capture-avoiding formulae φ, it holds that*

$$\varphi \equiv \varphi[i \mapsto (\mathsf{guard}^i(\varphi) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)}$$

Proof. By structural induction on the formula φ. The property $\varphi \equiv \varphi[i \mapsto (\mathsf{guard}^i(\varphi) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)}$ trivially holds for the base cases $\varphi = X(e)$, since $\mathsf{guard}^1(X(e)) = \mathit{true}$, and $\varphi = b$, since it does not contain any PVI. We distinguish four cases for the induction step. Since φ is normalised, we do not have to consider the case $\varphi = \neg\varphi_1$. Note that in none of the cases, $\mathsf{guard}^i(\varphi)$ contains a quantifier that does not exist in φ, so $\varphi[i \mapsto \mathsf{guard}^i(\varphi) \wedge \mathsf{PVI}(\varphi, i)]_{i \leq \mathsf{npred}(\varphi)}$ is also capture-avoiding. In each of the derivations below, $\star$ indicates that we use the fact that none of the involved substitutions affect the number or the identity of PVIs.

Case $\varphi = \varphi_1 \wedge \varphi_2$ We assume as induction hypothesis that $\varphi_1 \equiv \varphi_1[i \mapsto \mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi_1, i)]_{i \leq \mathsf{npred}(\varphi_1)}$ and $\varphi_2 \equiv \varphi_2[i \mapsto \mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi_2, i)]_{i \leq \mathsf{npred}(\varphi_2)}$. We perform the following derivation:

$$\begin{aligned}
&(\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&= (\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i}
\end{aligned}$$

$$
\begin{aligned}
&= (\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge s(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge s(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\overset{(\star)}{=} (\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (s(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\qquad [i \mapsto (s(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\overset{(\star)}{=} (\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\qquad [i \mapsto (s(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} [i \mapsto (s(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\overset{(\dagger)}{\equiv} (\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&= \varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad \wedge \varphi_2[i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_2)} \\
&\overset{(IH)}{\equiv} \varphi_1 \wedge \varphi_2
\end{aligned}
$$

At †, we apply Lemma 7.36 twice; remark that the body of the substitutions we remove is stronger than $\varphi_1 \wedge \varphi_2$ (by monotonicity of φ), so it certainly implies $s(\varphi_1)$ and $s(\varphi_2)$ by Lemma 7.35.

Case $\varphi = \varphi_1 \vee \varphi_2$ We assume as induction hypothesis that $\varphi_1 \equiv \varphi_1[i \mapsto \mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi_1, i)]_{i \leq \mathsf{npred}(\varphi_1)}$ and $\varphi_2 \equiv \varphi_2[i \mapsto \mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi_2, i)]_{i \leq \mathsf{npred}(\varphi_2)}$. We perform the following derivation:

$$
\begin{aligned}
&(\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1 \vee \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&= (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1 \vee \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_1 \vee \varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&= (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge s(\neg\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge s(\neg\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\overset{(\star)}{=} (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (s(\neg\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\qquad [i \mapsto (s(\neg\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\overset{(\star)}{=} (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&\qquad [i \mapsto (s(\neg\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} [i \mapsto (s(\neg\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i}
\end{aligned}
$$

$$
\begin{aligned}
&\overset{(\dagger)}{\equiv} (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad\qquad [i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i} \\
&= \varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\qquad\qquad \vee \varphi_2[i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\overset{(IH)}{\equiv} \varphi_1 \vee \varphi_2
\end{aligned}
$$

At †, we apply Lemma 7.37 twice. The assumption $\varphi' \to \varphi$ from the lemma holds, since $\varphi_1 \to \varphi_1 \vee \varphi_2 \equiv (\varphi_1 \vee \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)}[i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i}$ (and similarly for φ_2), by the induction hypothesis. Furthermore, the other assumptions of the lemma also hold, since

$$
\begin{aligned}
&\mathsf{vars}(s(\neg\varphi_1)) \subseteq \mathsf{vars}(\varphi_1) \subseteq \mathsf{vars}(\varphi_1 \vee \varphi_2) = \mathsf{vars}((\varphi_1 \vee \varphi_2) \\
&[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)}[i \mapsto (\mathsf{guard}^i(\varphi_2) \wedge \mathsf{PVI}(\varphi, i))]_{\mathsf{npred}(\varphi_1) < i})
\end{aligned}
$$

The same holds for φ_2.

Case $\varphi = \forall d{:}D.\, \varphi_1$ As induction hypothesis, we assume that $\varphi_1 \equiv \varphi_1[i \mapsto \mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi_1, i)]_{i \leq \mathsf{npred}(\varphi_1)}$. We derive:

$$
\begin{aligned}
&(\forall d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\forall d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&= (\forall d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge s(\forall d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\overset{(\star)}{=} (\forall d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\qquad\qquad [i \mapsto (s(\forall d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\overset{(\dagger)}{\equiv} (\forall d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&= \forall d{:}D.\, \varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\
&\overset{(IH)}{\equiv} \forall d{:}D.\, \varphi_1
\end{aligned}
$$

At †, we can apply Lemma 7.36 since we have

$$(\forall d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \to \forall d{:}D.\, \varphi_1 \to s(\forall d{:}D.\, \varphi_1)$$

which follows from monotonicity of φ_1 and Lemma 7.35.

Case $\varphi = \exists d{:}D.\, \varphi_1$ For the induction hypothesis, we assume that $\varphi_1 \equiv \varphi_1[i \mapsto \mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi_1, i)]_{i \leq \mathsf{npred}(\varphi_1)}$. We derive:

$$
\begin{aligned}
&(\exists d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\exists d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&= (\exists d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge s(\neg\exists d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\overset{(\star)}{=} (\exists d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\qquad\qquad [i \mapsto (s(\neg\exists d{:}D.\, \varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)} \\
&\overset{(\dagger)}{\equiv} (\exists d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)}
\end{aligned}
$$

$$\begin{aligned} &= \exists d{:}D.\, \varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)} \\ &\overset{(IH)}{\equiv} \exists d{:}D.\, \varphi_1 \end{aligned}$$

At †, we apply Lemma 7.37. Its assumption $\varphi' \to \varphi$ is valid since

$$\exists d{:}D.\, \varphi_1 \to (\exists d{:}D.\, \varphi_1)[i \mapsto (\mathsf{guard}^i(\varphi_1) \wedge \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi_1)}$$

holds by the induction hypothesis. Furthermore, we have $\mathsf{vars}(s(\neg\exists d{:}D.\, \varphi)) \subseteq \mathsf{vars}(\exists d{:}D.\, \varphi_1)$. □

Remark that the assumption that φ is capture-avoiding is necessary for the correctness of the guard function. Without this assumption, we may compute $\mathsf{guard}^1((n \leq 2) \wedge \forall n{:}N.\, X(n)) = (n \leq 2)$, which is not a guard (see also Example 7.30). This predicate formula is also a counter-example to the correctness of [73, Lemma 6.27], which is the counterpart of Theorem 7.38 in the current work. Adding the assumption that φ is capture-avoiding resolves the issue.

7.6.3 Exact Guards

The stronger the guard that we compute, the more information can be extracted from it. Ideally, we would like to compute the strongest formula that is a guard. We call this the *exact guard*: the strongest formula ψ such that $\varphi \equiv \varphi[i \mapsto \psi \wedge \mathsf{PVI}(\varphi, i)]$. Example 7.32 already showed that guards are not necessarily compositional. The same example also shows that exact guards are not compositional, since *false* is an exact guard for both PVIs in $X \vee X$. The next lemma aims to provide some intuition on why the guard function comes close to the exact guard.

Lemma 7.39. *For all monotone φ, $s(\varphi)$ is the strongest simple formula χ such that $\varphi \to \chi$,* i.e.*, for all simple ψ, $\varphi \to \psi$ implies $s(\varphi) \to \psi$. Dually, $s(\neg\varphi)$ is the strongest simple formula χ' such that $\neg\varphi \to \chi'$,* i.e.*, for all simple ψ, $\neg\varphi \to \psi$ implies $s(\neg\varphi) \to \psi$.*

Proof. Here, we discuss the case of $s(\varphi)$, the other case is dual. Let φ be some monotone formula and let η and δ be arbitrary. Furthermore, let η_{true} be such that $v \in \eta_{true}(X)$ for all $X \in \mathsf{occ}(\varphi)$ and $v \in \mathbb{D}$. The implication $\varphi \to s(\varphi)$ follows from Lemma 7.35.

To prove that $s(\varphi)$ is the strongest simple formula with this property, let ψ be some simple formula such that $[\![\varphi]\!]\eta\delta \Rightarrow [\![\psi]\!]\eta\delta$. Then, in the particular case that $\eta = \eta_{true}$, it holds $[\![\varphi]\!]\eta_{true}\delta \Rightarrow [\![\psi]\!]\eta\delta$. With $[\![\varphi]\!]\eta_{true}\delta = [\![s(\varphi)]\!]\eta\delta$, we conclude that $[\![s(\varphi)]\!]\eta\delta \Rightarrow [\![\psi]\!]\eta\delta$. □

7.7 Implementation

The proposed generalisations for constant elimination and guards have been implemented as an extension of the tool `pbesconstelm`, which implements constant

elimination and is part of the mCRL2 toolset [26]. Before computing the quantified constants with the fixpoint algorithm of Section 7.5, all guards and occurrences of QPVIs are first obtained by recursively traversing each right-hand side in the PBES. Furthermore, we gather information on the free occurrences of variables, such that the effect of the quantifier-inside rewriter can be approximated. Hence, no traversal of the right-hand side is required during the fixpoint computation itself.

To demonstrate a possible application of our ideas, we first perform a small experiment with a model of the *alternating bit protocol* (ABP), where the data domain D has been restricted to only five elements. The LTS of this model has 182 states. We consider the property

$$\forall d{:}D.\, \nu W.\, \big([\top]W \wedge \nu X.\, \mu Y.\, \nu Z.\, ([r(d)]X \wedge (\langle r(d)\rangle true \Rightarrow [\overline{r(d)}]Y) \wedge [\overline{r(d)}]Z\big)$$

which expresses that whenever $r(d)$ is enabled infinitely often, then it also taken infinitely often, *i.e.*, it is treated fairly. This formula occurred earlier in [73]. Observe that the universally quantified value d does not occur meaningfully in the fixpoint W. We thus expect that the same quantifier in the corresponding PBES can be eliminated.

We take the PBES that encodes this formula on the ABP and instantiate it to a parity game. This parity game has 3641 nodes. Applying the original constant elimination algorithm from [106] on the PBES does not reduce this number. After applying our generalised algorithm that deals with quantifiers, the instantiated parity game is reduced to 2913 nodes.

Our second experiment concerns the *cache coherence protocol* (CCP), as modelled in [108]. The instance we consider has two threads, two processes and one region. We slightly altered the specification, such that process identifiers are modelled with natural number instead of a dedicated finite sort; this does not change its behaviour. We consider the property that, if the system is stable, each region has no more copies than the number of processors minus one, formulated in [73] as

$$\begin{aligned}\forall procId{:}N.\, \neg\big(\mu X.\, \langle true\rangle \vee (\langle c\_copy\rangle true \wedge \langle lockempty(procId)\rangle true \\ \wedge \langle homequeueempty(procId)\rangle true \wedge \langle remotequeueempty(procId)\rangle true)\big)\end{aligned}$$

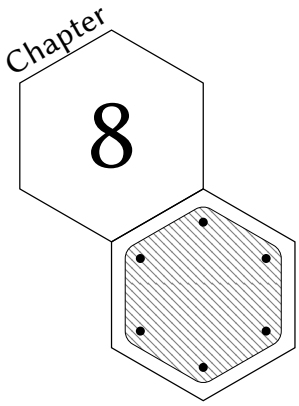
The corresponding PBES contains a QPVI of the shape $\forall procId{:}N.\, X(d, procId)$ and has an underlying parity game of infinite size; the PBES can hence not be instantiated. However, after applying global propagation, the size of the parity game is reduced to 50507 nodes. Global propagation also achieves a reduction when applied to the original specification, where the sort of variable *procId* contains only two elements. In that case, the size of the parity game is reduced from 101197 to 50507 nodes.

7.8 Conclusion

In this chapter, we saw that quantifiers can be a cause of state space explosion in PBESs. However, quantifiers have barely received attention in the literature on

syntactic PBES transformations. Hence, we proposed quantifier propagation and global propagation, which can achieve state space reduction by syntactic manipulation of the quantifiers in a PBES. Our experiment with global propagation indicates that it can indeed achieve a reduction of the state space. Furthermore, we identified an improvement for the computation of guards.

The propagation techniques proposed in this chapter can only deal with quantifiers that are not accompanied by a bound, *i.e.*, expressions of the shape $\mathsf{Q}d{:}D.\, X(e)$. When adding bounds to the quantification, by writing $\forall d{:}D.\, f(d) \Rightarrow X(e)$ or $\exists d{:}D.\, f(d) \wedge X(e)$, neither technique has any effect. We believe the support of quantifier bounds will greatly benefit its practical applicability, so we plan to handle these cases in future work. We also aim to improve the definition of guard, such that it yields a natural notion of exact guard that is compositional.



Conclusion

We have studied the theory of μ-calculus model checking through parity games and parameterised Boolean equation systems. Although the use of PBESs has several advantages, it does not immediately solve the state explosion problem that is pervasive in model checking. Consequently, the parity game that is encoded in a PBES might be prohibitively large, preventing straightforward solving techniques based on instantiation. Therefore, we investigated the application of several reduction techniques to PBESs.

8.1 Summary

In Chapter 3 we reviewed several normal forms for PBESs, and proposed two new normal forms: SRF and CRF. They allow (symbolic) reasoning about the transitions in the dependency graph and, because alternations between conjunctions and disjunctions have been eliminated, to construct a dependency space. The dependency space contains both positive and negative dependencies, and thus captures sufficient information to solve the PBES. Since dependency spaces coincide with parity games, we can apply existing fundamental theory for parity games to (symbolically) solve PBESs.

The normal forms are first applied in Chapter 4, which proposes the symbolic

technique PBES quotienting. This semi-decision procedure attempts to compute the bisimulation quotient of the associated parity game by incrementally refining a partition of the state space. The application on PBESs and parity games allows two optimisations that are not possible in the classical LTS setting. In an experimental evaluation, our implementation of PBES quotienting is able to solve more problem instances than existing tools. Furthermore, PBES quotienting can be applied as a semi-decision procedure for (branching) bisimulation on infinite-state systems; to our knowledge, there is no other fully automated tool that can solve these equivalence problems.

Next, Chapter 5 introduced the existing theory of partial-order reduction for LTL_{-X} model checking and shows that, contrary to what is stated in earlier works, stutter trace equivalence is not necessarily preserved. This is called the inconsistent labelling problem. It can be remedied by strengthening one of the reduction conditions. We analysed several settings and identified in which cases the inconsistent labelling problem may occur: the theory of at least four related works is affected. Fortunately, the practical implications seem limited: all implementations approximate the reduction based on stronger conditions, thereby avoiding the problem.

We applied the updated conditions in the setting of PBESs and parity games in Chapter 6. The correctness proof is non-trivial: it does not automatically follow from preservation of stutter trace equivalence of the LSTS that corresponds to a parity game. Here, we again relied on the SRF normal form to reason about the associated parity game of a PBES. Experimental results show that POR can achieve significant reductions; a reduction of more than 90% is not uncommon.

Finally, Chapter 7 discussed the topic of syntactical transformations on PBESs that aim to reduce the size of the underlying parity game. We proposed a generalisation of the constant elimination technique for PBESs that also involves quantifiers. Our implementation is able to transform a PBES, such that its underlying graph structure becomes finite, and the PBES can be solved through instantiation. Furthermore, we showed how to compute better guards for predicate formulae, a fundamental tool for many static analysis techniques.

8.2 Discussion

We reflect on the ideas presented in the thesis, and discuss them in the context of the question whether the application of existing reduction techniques on PBESs has significant advantages or disadvantages, when compared to their equivalent technique for LPSs.

The main benefit of model checking with PBESs is that they allow more powerful abstractions than LPSs. After all, a PBES encodes only a single property, thus we only have to preserve that property. We have seen several examples that exploit this fact, they show that our PBES quotienting and POR techniques are more powerful than their equivalent LPS techniques. Our comparative experiments in Chapter 4 also confirm this intuition: we are able to check several properties through PBES

quotienting that could not be checked with minimal model generation. The quantifier manipulation techniques of Chapter 7 do not have a clear equivalent in the setting of LPSs, again illustrating that PBESs are more versatile than LPSs. A further benefit of the PBES model checking approach is that the reduction techniques we have developed can potentially also be used for other types of decision problems that can be encoded in a PBES.

However, the application of PBESs also has drawbacks. First of all, in case one wants to check multiple properties on the same LPS, we obtain an equal number of PBESs that each have to be solved individually. This may cause a lot of repetitive work. For LPSs where one can efficiently compute an LTS that accurately represent its semantics, *e.g.*, with state-space exploration or minimal model generation (see Chapter 4), it can be more efficient to perform model checking on that LTS.

Secondly, the fact that PBESs are widely applicable also implies that it can be difficult to extract sufficient information on the underlying problem. For example, in the case of model checking, it is sometimes desirable to know whether (a part of) the state space is encoded twice in two equations or which parameters belong to the same concurrent process. In an SRF-PBES, the relation between processes and clauses is another piece of valuable information that is not easy to recover. This is one of the main challenges in Chapter 6: since no information on the embedded concurrent processes and their actions is available in a PBES, it is difficult to determine a good edge labelling function. Moreover, a significant amount of static analysis is required to construct the accordance relations. Do note, however, that LPSs are not free from these problems: in an LPS, the structure of processes is also mostly lost. For PBESs, a possible solution is to encode the necessary information in the PBES, in a way that does not affect its solution. This approach is applied in [136], which shows how to extract a model checking witness/counter-example (in the form of an LPS) after a PBES has been solved.

8.3 Future Work

The correctness proofs for PBES transformation techniques can be very tricky, especially when dealing with arbitrary PBESs. Chapter 7 contains several such proofs. This problem was also identified by Willemse [137], who developed the notion of *consistent correlation* with the goal of simplifying such proofs. A consistent correlation is an equivalence relation on nodes $X(e)$ in a PBES; nodes that are related by some consistent correlation have the same solution. By relating nodes in the original PBES and the transformed PBES, one can prove that a transformation is solution-preserving. However, consistent correlation is strongly related to idempotence-identifying bisimulation [75]; these notions are equivalent for BESs in SRF [137]. Thus, consistent correlation is only suited for proving correctness of transformations that preserve bisimulation. Consistent consequence [34], a similar notion that coincides with simulation on SRF-BESs, only partially resolves this shortcoming. Some of the techniques we studied preserve only stutter trace equivalence, which is a weaker

notion than bisimulation and simulation equivalence. Developing a theory that is similar to consistent correlation, but based on trace equivalence, can significantly simplify the correctness proofs of these techniques.

As discussed in the previous section, extracting information from arbitrary PBESs is difficult. At the same time, this is a cornerstone of many syntactic transformation techniques. Thus, future research should focus on refining the existing techniques, such as the guards of Section 7.6 and the control flow analysis of [74], and exploring new concepts. Here, one may take inspiration from the analysis techniques embedded in many modern compilers, which require this information for the application of optimisations. The information obtained through static analysis may also benefit other techniques. For example, information on the control flow graph encoded in a PBES may be used to construct a finer initial partition in the symbolic approach of Chapter 4 or to speed up the static analysis for the POR technique of Chapter 6.

One way to tackle the problem of the choice of labelling function in Chapter 6, is to generalise the POR theory so it is agnostic of the edge labelling. Here, one can draw inspiration from the related theory of confluence reduction [57], where all invisible actions are called τ. Then, the generalisation requires that, instead of choosing a subset of action labels in each state, one chooses a subset of the outgoing τ-transitions. A challenge in such a theory is ensuring that the accompanying implementation is still efficient.

A major obstacle for the practical application of both the symbolic bisimulation procedures of Chapter 4 and the POR algorithm of Chapter 6 is the reasoning about predicate formulas containing infinite algebraic data types, such as lists. While SMT solvers are relatively strong when it comes to deciding satisfiability within theories over integers, reals or finite arrays, the ability to reason about algebraic data types (ADTs) is limited. A short experiment with SMT-queries containing lists, produced by `pbespor`, revealed that CVC4 [10] also cannot decide satisfiability of all these expressions, even though it includes several techniques aimed at ADTs [11]. Thus, more research into deciding ADT theories is required. The ability to decide most expressions over first-order ADTs, such as lists, can already greatly improve the practical applicability of our PBES solving techniques.

Bibliography

[1] Software Engineering: Report on a Conference Sponsored by the NATO Science Committe. p. 136, NATO, 1968.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi, Minimization of Timed Transition Systems. In *CONCUR 1992*, vol. 630 of *LNCS*, pp. 340–354, 1992, DOI: 10.1007/BFb0084777.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, Hybrid automata: An algorithmic Approach to the Specification and Verification of Hybrid Systems. In *HS 1992, HS 1991*, vol. 736 of *LNCS*, pp. 209–229, 1993, DOI: 10.1007/3-540-57318-6_30.

[4] R. Alur and D. L. Dill, A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994, DOI: 10.1016/0304-3975(94)90010-8.

[5] T. E. Anderson, The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):6–16, 1990, DOI: 10.1109/71.80120.

[6] J. C. M. Baeten, A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005, DOI: 10.1016/j.tcs.2004.07.036.

[7] J. C. M. Baeten and W. P. Weijland, *Process Algebra*, vol. 18 of *Cambridge tracts in theoretical computer science*. 1990.

[8] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Algebraic decision diagrams and their applications. In *ICCAD 1993*, pp. 188–191, IEEE, 1993, DOI: 10.1109/ICCAD.1993.580054.

[9] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT Press, 2008.

[10] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovi, T. King, A. Reynolds, and C. Tinelli, CVC4. In *CAV 2011*, vol. 6806 of *LNCS*, pp. 171–177, 2011, DOI: 10.1007/978-3-642-22110-1_14.

[11] C. Barrett, I. Shikanian, and C. Tinelli, An Abstract Decision Procedure for a Theory of Inductive Data Types. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(1-2):21–46, 2007, DOI: 10.3233/SAT190028.

[12] G. Behrmann, A. David, and K. G. Larsen, A Tutorial on UPPAAL. In *SFM-RT 2004*, vol. 3185 of *LNCS*, pp. 200–236, 2004, DOI: 10.1007/978-3-540-30080-9_7.

[13] H. Bekič, Definable operations in general algebras, and the theory of automata and flowcharts. In *Programming Languages and Their Definition*, LNCS, pp. 30–55, 1984, DOI: 10.1007/BFb0048939.

[14] N. Beneš, L. Brim, B. Buhnova, I. Ern, J. Sochor, and P. Vařeková, Partial order reduction for state/event LTL with application to component-interaction automata. *Science of Computer Programming*, 76(10):877–890, 2011, DOI: 10.1016/j.scico.2010.02.008.

[15] N. Beneš, L. Brim, I. Černá, J. Sochor, P. Vařeková, and B. Zimmerova, Partial Order Reduction for State/Event LTL. In *IFM 2009*, vol. 5423 of *LNCS*, pp. 307–321, 2009, DOI: 10.1007/978-3-642-00255-7_21.

[16] J. A. Bergstra and J. W. Klop, Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984, DOI: 10.1016/S0019-9958(84)80025-X.

[17] J. A. Bergstra and J. W. Klop, Verification of an alternating bit protocol by means of process algebra protocol. In *MMSSS 1985*, vol. 215 of *LNCS*, pp. 9–23, 1986, DOI: 10.1007/3-540-16444-8_1.

[18] G. Bhat and R. Cleaveland, Efficient model checking via the equational μ-calculus. In *LICS 1996*, pp. 304–312, 1996, DOI: 10.1109/LICS.1996.561358.

[19] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, Symbolic model checking without BDDs. In *TACAS 1999*, vol. 1579 of *LNCS*, pp. 193–207, 1999, DOI: 10.1007/3-540-49059-0_14.

[20] F. M. Bønneland, P. G. Jensen, K. G. Larsen, and M. Muñiz, Partial Order Reduction for Reachability Games. In *CONCUR 2019*, vol. 140, pp. 23:1–23:15, 2019, DOI: 10.4230/LIPIcs.CONCUR.2019.23.

[21] F. M. Bønneland, P. G. Jensen, K. G. Larsen, M. Mũniz, and J. Srba, Stubborn Set Reduction for Two-Player Reachability Games. arXiv:1912.09875, 2019.

[22] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs, Minimal Model Generation. In *CAV 1990*, vol. 531 of *LNCS*, pp. 197–203, 1990, DOI: 10.1007/BFb0023733.

[23] A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel, Minimal state graph generation. *Science of Computer Programming*, 18(3):247–269, 1992, DOI: 10.1016/0167-6423(92)90018-7.

[24] J. Bradfield and I. Walukiewicz, The mu-calculus and Model Checking. In *Handbook of Model Checking*, pp. 871–919, 2018, DOI: 10.1007/978-3-319-10575-8_26.

[25] R. E. Bryant, Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, 1986, DOI: 10.1109/TC.1986.1676819.

[26] O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, W. Wesselink, A. Wijs, and T. A. C. Willemse, The mCRL2 Toolset for Analysing Concurrent Systems: Improvements in Expressivity and Usability. In *TACAS 2019*, vol. 11428 of *LNCS*, pp. 21–39, 2019, DOI: 10.1007/978-3-030-17465-1_2.

[27] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, Symbolic model checking: 10^{20} states and beyond. In *LICS 1990*, pp. 428–439, IEEE, 1990, DOI: 10.1109/LICS.1990.113767.

[28] T. Chen, B. Ploeger, J. van de Pol, and T. A. C. Willemse, Equivalence Checking for Infinite Systems using Parameterized Boolean Equation Systems. In *CONCUR 2007*, vol. 4703 of *LNCS*, pp. 120–135, 2007, DOI: 10.1007/978-3-540-74407-8_9.

[29] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, Counterexample-Guided Abstraction Refinement. In *CAV 2000*, vol. 1855 of *LNCS*, pp. 154–169, 2000, DOI: 10.1109/TIME.2003.1214874.

[30] E. M. Clarke and E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs 1981*, vol. 131 of *LNCS*, pp. 52–71, 1982, DOI: 10.1007/BFb0025774.

[31] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. Yang, Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping. In *DAC 1993*, pp. 54–60, IEEE, 1993, DOI: 10.1145/157485.164569.

[32] P. Cousot and R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL 1977*, pp. 238–252, 1977, DOI: 10.1145/512950.512973.

[33] S. Cranen, M. Gazda, W. Wesselink, and T. A. C. Willemse, Abstraction in Fixpoint Logic. *ACM Transactions on Computational Logic*, 16(4):Article 29, 2015, DOI: 10.1145/2740964.

[34] S. Cranen, M. W. Gazda, J. W. Wesselink, and T. A. C. Willemse, Abstraction in Parameterised Boolean Equation Systems. Tech. rep., Technische Universiteit Eindhoven, 2013.

[35] S. Cranen, J. F. Groote, and M. Reniers, A linear translation from CTL* to the first-order modal μ-calculus. *Theoretical Computer Science*, 412(28):3129–3139, 2011, DOI: 10.1016/j.tcs.2011.02.034.

[36] S. Cranen, J. J. A. Keiren, and T. A. C. Willemse, A Cure for Stuttering Parity Games. In *ICTAC 2012*, vol. 7521 of *LNCS*, pp. 198–212, 2012, DOI: 10.1007/978-3-642-32943-2_16.

[37] S. Cranen, B. Luttik, and T. A. C. Willemse, Proof graphs for parameterised Boolean equation systems. In *CONCUR 2013*, vol. 8052 of *LNCS*, pp. 470–484, 2013, DOI: 10.1007/978-3-642-40184-8_33.

[38] L. De Moura and N. Bjørner, Z3: An efficient SMT Solver. In *TACAS 2008*, vol. 4963 of *LNCS*, pp. 337–340, 2008, DOI: 10.1007/978-3-540-78800-3_24.

[39] E. W. Dijkstra, The Humble Programmer. *Communications of the ACM*, 15(10):859–866, 1972, DOI: 10.1145/355604.361591.

[40] C. Ebert and J. Capers, Embedded Software: Facts, Figure and Future. *Computer*, 42(4):42–52, 2009, DOI: 10.1109/MC.2009.118.

[41] E. A. Emerson and J. Y. Halpern, "Sometimes" and "not never" revisited: on branching versus linear time. In *POPL 1983*, pp. 127–140, ACM Press, 1983, DOI: 10.1145/567067.567081.

[42] E. A. Emerson, S. Jha, and D. Peled, Combining Partial Order and Symmetry Reductions. In *TACAS 1997*, vol. 1217 of *LNCS*, pp. 19–34, 1997, DOI: 10.1007/BFb0035378.

[43] E. A. Emerson and C. S. Jutla, Tree automata, mu-calculus and determinacy. In *FOCS 1991*, pp. 368–377, 1991, DOI: 10.1109/sfcs.1991.185392.

[44] S. Evangelista and C. Pajault, Solving the ignoring problem for partial order reduction. *International Journal on Software Tools for Technology Transfer*, 12:155–170, 2010, DOI: 10.1007/s10009-010-0137-y.

[45] J. C. Filliâtre, Deductive software verification. *International Journal on Software Tools for Technology Transfer*, 13(5):397–403, 2011, DOI: 10.1007/s10009-011-0211-0.

[46] K. Fisler and M. Y. Vardi, Bisimulation and model checking. In *CHARME 1999*, vol. 1703 of *LNCS*, pp. 338–342, 1999, DOI: 10.1007/3-540-48153-2_29.

[47] P. Fontana and R. Cleaveland, The Power of Proofs: New Algorithms for Timed Automata Model Checking. In *FORMATS 2014*, vol. 8711 of *LNCS*, pp. 115–129, 2014, DOI: 10.1007/978-3-319-10512-3_9.

[48] J. B. J. Fourier, Solution d'une question particuliere du calcul des inégalités. *Nouveau Bulletin des Sciences par la Société Philomatique de Paris*, 99:100, 1826.

[49] M. W. Gazda and T. A. C. Willemse, On Parity Game Preorders and the Logic of Matching Plays. In *SOFSEM 2016*, vol. 9587 of *LNCS*, pp. 277–289, 2016, DOI: 10.1007/978-3-662-49192-8_23.

[50] R. Gerth, R. Kuiper, D. Peled, and W. Penczek, A Partial Order Approach to Branching Time Logic Model Checking. *Information and Computation*, 150(2):132–152, 1999, DOI: 10.1006/inco.1998.2778.

[51] T. Gibson-Robinson, H. Hansen, A. W. Roscoe, and X. Wang, Practical Partial Order Reduction for CSP. In *NFM 2015*, vol. 9058 of *LNCS*, pp. 188–203, 2015, DOI: 10.1007/978-3-319-17524-9_14.

[52] P. Godefroid, *Partial-Order Methods for the Verification of Concurrent Systems*, vol. 1032 of *LNCS*. Springer, 1996, DOI: 10.1007/3-540-60761-7.

[53] E. Goldberg and P. Manolios, Partial quantifier elimination. In *HVC 2014*, vol. 8855 of *LNCS*, pp. 148–164, 2014, DOI: 10.1007/978-3-319-13338-6_12.

[54] E. Grädel and I. Walukiewicz, Positional Determinacy of Games with Infinitely Many Priorities. *Logical Methods in Computer Science*, 2(4):1–22, 2006, DOI: 10.2168/LMCS-2(4:6)2006.

[55] J. F. Groote and R. Mateescu, Verification of temporal properties of processes in a setting with data. In *AMAST 1998*, vol. 1548 of *LNCS*, pp. 74–90, 1998, DOI: 10.1007/3-540-49253-4_8.

[56] J. F. Groote and M. R. Mousavi, *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[57] J. F. Groote and M. P. A. Sellink, Confluence for process verification. *Theoretical Computer Science*, 170(1-2):47–81, 1996, DOI: 10.1016/s0304-3975(96)00175-2.

[58] J. F. Groote and T. A. C. Willemse, Model-checking processes with data. *Science of Computer Programming*, 56(3):251–273, 2005, DOI: 10.1016/j.scico.2004.08.002.

[59] J. F. Groote and T. A. C. Willemse, Parameterised boolean equation systems. *Theoretical Computer Science*, 343(3):332–369, 2005, DOI: 10.1016/j.tcs.2005.06.016.

[60] O. Grumberg and D. E. Long, Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994, DOI: 10.1145/177492.177725.

[61] R. Hähnle and M. Huisman, Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In *Computing and Software Science*, vol. 2 of *LNCS*, pp. 345–373, 2019, DOI: 10.1007/978-3-319-91908-9_18.

[62] H. Hansen, S.-W. Lin, Y. Liu, T. K. Nguyen, and J. Sun, Diamonds Are a Girl's Best Friend: Partial Order Reduction for Timed Automata with Abstractions. In *CAV 2014*, vol. 8559 of *LNCS*, pp. 391–406, 2014, DOI: 10.1007/978-3-319-08867-9_26.

[63] D. Heimbold and D. Luckham, Debugging Ada Tasking Programs. *IEEE Software*, 2(2):47–57, 1985, DOI: 10.1109/MS.1985.230351.

[64] M. Hennessy and R. Milner, On observing nondeterminism and concurrency. In *ICALP 1980*, vol. 85 of *LNCS*, pp. 299–309, 1980, DOI: 10.1007/3-540-10003-2_79.

[65] W. H. Hesselink, Invariants for the construction of a handshake register. *Information Processing Letters*, 68(4):173–177, 1998, DOI: 10.1016/s0020-0190(98)00158-6.

[66] M. Hutagalung and M. Lange, Model checking for string problems. In *CSR 2014*, vol. 8476 of *LNCS*, pp. 190–203, 2014, DOI: 10.1007/978-3-319-06686-8_15.

[67] C. N. Ip and D. L. Dill, Better verification through symmetry. *Formal Methods in System Design*, 9(1-2):41–75, 1996, DOI: 10.1007/BF00625968.

[68] M. Jurdziński, Deciding the winner in parity games is in UP and co-UP. *Information Processing Letters*, 68(3):119–124, 1998, DOI: 10.1016/s0020-0190(98)00150-1.

[69] A. Kaldewaij, *Programming: The derivation of algorithms*. Prentice Hall, 1990.

[70] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, Multi-Valued Decision Diagrams: Theory and Applications. *Multiple-Valued Logic*, 4(1-2):9–62, 1998.

[71] S. Kan, Z. Huang, Z. Chen, W. Li, and Y. Huang, Partial order reduction for checking LTL formulae with the next-time operator. *Journal of Logic and Computation*, 27(4):1095–1131, 2017, DOI: 10.1093/logcom/exw004.

[72] G. Kant and J. van de Pol, Efficient Instantiation of Parameterised Boolean Equation Systems to Parity Games. In *GRAPHITE 2012*, vol. 99 of *EPTCS*, pp. 50–65, 2012, DOI: 10.4204/EPTCS.99.7.

[73] J. J. A. Keiren, *Advanced Reduction Techniques for Model Checking*. Ph.D. thesis, Eindhoven University of Technology, 2013, DOI: 10.6100/IR757862.

[74] J. J. A. Keiren, W. Wesselink, and T. A. C. Willemse, Liveness Analysis for Parameterised Boolean Equation Systems. In *ATVA 2014*, vol. 8837 of *LNCS*, pp. 219–234, 2014, DOI: 10.1007/978-3-319-11936-6_16.

[75] J. J. A. Keiren and T. A. C. Willemse, Bisimulation minimisations for Boolean equation systems. In *HVC 2009*, vol. 6405 of *LNCS*, pp. 102–116, 2011, DOI: 10.1007/978-3-642-19237-1_12.

[76] R. M. Keller, Formal Verification of Parallel Programs. *Communications of the ACM*, 19(7):371–384, 1976, DOI: 10.1145/360248.360251.

[77] D. Kőnig, Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3(2-3):121–130, 1927.

[78] B. Knaster, Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématique*, 6:133–134, 1928.

[79] D. E. Knuth, Textbook Examples of Recursion. *Artificial and Mathematical Theory of Computation*, 91:207–229, 1991, DOI: 10.1016/B978-0-12-450010-5.50018-9.

[80] N. Kobayashi, T. Nishikawa, A. Igarashi, and H. Unno, Temporal Verification of Programs via First-Order Fixpoint Logic. In *SAS 2019*, vol. 11822 of *LNCS*, pp. 413–436, 2019, DOI: 10.1007/978-3-030-32304-2_20.

[81] R. P. J. Koolen, T. A. C. Willemse, and H. Zantema, Using SMT for Solving Fragments of Parameterised Boolean Equation Systems. In *ATVA 2015*, vol. 9364 of *LNCS*, pp. 14–30, 2015, DOI: 10.1007/978-3-319-24953-7_3.

[82] C. P. J. Koymans and J. C. Mulder, A modular approach to protocol verification using process algebra. In *Applications of Process Algebra*, pp. 261–306, sep 1990, DOI: 10.1017/CBO9780511608841.012.

[83] D. Kozen, Results on the propositional μ-calculus. In *ICALP 1982*, vol. 140 of *LNCS*, pp. 348–359, 1982, DOI: 10.1007/BFb0012782.

[84] D. Kozen, Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983, DOI: 10.1016/0304-3975(82)90125-6.

[85] A. Laarman, E. Pater, J. van de Pol, and H. Hansen, Guard-based partial-order reduction. *International Journal on Software Tools for Technology Transfer*, 18(4):427–448, 2016, DOI: 10.1007/s10009-014-0363-9.

[86] L. Lamport, A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974, DOI: 10.1145/361082.361093.

[87] L. Lamport, A Fast Mutual Exclusion Algorithm. *ACM Transactions on Computer Systems*, 5(1):1–11, 1987, DOI: 10.1145/7351.7352.

[88] G. Le Lann, Distributed Systems - Towards a Formal Aproach. In *IFIP Congress*, pp. 155–160, 1977.

[89] C. Y. Lee, Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technical Journal*, 38(4):985–999, 1959, DOI: 10.1002/j.1538-7305.1959.tb01585.x.

[90] D. Lee and M. Yannakakis, Online minimization of transition systems (extended abstract). In *STOC '92*, pp. 264–274, 1992, DOI: 10.1145/129712.129738.

[91] T. Liebke and K. Wolf, Taking Some Burden Off an Explicit CTL Model Checker. In *Petri Nets 2019*, vol. 11522 of *LNCS*, pp. 321–341, 2019, DOI: 10.1007/978-3-030-21571-2_18.

[92] A. Mader, Modal μ-calculus, model checking and Gauß elimination. In *TACAS 1995*, vol. 1019 of *LNCS*, pp. 72–88, 1995, DOI: 10.1007/3-540-60630-0_4.

[93] A. Mader, *Verification of Modal Properties using Boolean Equation Systems*. Ph.D. thesis, Technische Universität München, 1997.

[94] A. A. Markov, An Example of Statistical Investigation of the Text *Eugene Onegin* Concerning the Connection of Samples in Chains. *Science in Context*, 19(4):591–600, 2006, DOI: 10.1017/S0269889706001074, translated from Russian by Gloria Custance and David Link.

[95] R. Mateescu, *Vérification des propriétés temporelles des programmes parallèles*. Ph.D. thesis, Institut National Polytechnique de Grenoble, 1998.

[96] R. Mateescu and A. Wijs, Property-dependent reductions adequate with divergence-sensitive branching bisimilarity. *Science of Computer Programming*, 96(P3):354–376, 2014, DOI: 10.1016/j.scico.2014.04.004.

[97] R. McNaughton, Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993, DOI: 10.1016/0168-0072(93)90036-D.

[98] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *LNCS*. 1980, DOI: 10.1007/3-540-10235-3.

[99] T. S. Motzkin, *Beiträge zur Theorie der linearen Ungleichungen*. Ph.D. thesis, Universität Basel, 1936.

[100] Y. Nagae and M. Sakai, Reduced Dependency Spaces for Existential Parameterised Boolean Equation Systems. In *WPTE 2017*, vol. 265 of *EPTCS*, pp. 67–81, 2017, DOI: 10.4204/EPTCS.265.6.

[101] Y. Nagae, M. Sakai, and H. Seki, An Extension of Proof Graphs for Disjunctive Parameterised Boolean Equation Systems. In *WPTE 2016*, vol. 235 of *EPTCS*, pp. 46–61, 2016, DOI: 10.4204/EPTCS.235.4.

[102] T. Neele, A. Valmari, and T. A. C. Willemse, The Inconsistent Labelling Problem of Stutter-Preserving Partial-Order Reduction. In *FoSSaCS 2020*, vol. 12077 of *LNCS*, pp. 482–501, 2020, DOI: 10.1007/978-3-030-45231-5_25.

[103] T. Neele, T. A. C. Willemse, and J. F. Groote, Solving Parameterised Boolean Equation Systems with Infinite Data Through Quotienting. In *FACS 2018*, vol. 11222 of *LNCS*, pp. 216–236, 2018, DOI: 10.1007/978-3-030-02146-7_11.

[104] T. Neele, T. A. C. Willemse, and J. F. Groote, Finding compact proofs for infinite-data parameterised Boolean equation systems. *Science of Computer Programming*, 188:102389, 2020, DOI: 10.1016/j.scico.2019.102389.

[105] T. Neele, T. A. C. Willemse, and W. Wesselink, Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems. In *TACAS 2020*, vol. 12079 of *LNCS*, pp. 307–324, 2020, DOI: 10.1007/978-3-030-45231-5_25.

[106] S. Orzan, W. Wesselink, and T. A. C. Willemse, Static Analysis Techniques for Parameterised Boolean Equation Systems. In *TACAS 2009*, vol. 5505 of *LNCS*, pp. 230–245, 2009, DOI: 10.1007/978-3-642-00768-2_22.

[107] S. Orzan and T. A. C. Willemse, Invariants for Parameterised Boolean Equation Systems. *Theoretical Computer Science*, 411(11-13):1338–1371, 2010, DOI: 10.1016/j.tcs.2009.11.001.

[108] J. Pang, W. Fokkink, R. Hofman, and R. Veldema, Model checking a cache coherence protocol of a Java DSM implementation. *Journal of Logic and Algebraic Programming*, 71(1):1–43, 2007, DOI: 10.1016/j.jlap.2006.08.007.

[109] D. Park, Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, vol. 104 of *LNCS*, pp. 167–183, 1981, DOI: 10.1007/BFb0017309.

[110] R. Pelánek, BEEM: Benchmarks for Explicit Model Checkers. In *SPIN 2007*, vol. 4595 of *LNCS*, pp. 263–267, 2007, DOI: 10.1007/978-3-540-73370-6_17.

[111] D. Peled, All from One, One for All: on Model Checking Using Representatives. In *CAV 1993*, vol. 697 of *LNCS*, pp. 409–423, 1993, DOI: 10.1007/3-540-56922-7_34.

[112] D. Peled, Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996, DOI: 10.1007/BF00121262.

[113] C. A. Petri, *Kommunikation mit Automaten*. Ph.D. thesis, Universität Bonn, 1962.

[114] B. Ploeger, J. W. Wesselink, and T. A. C. Willemse, Verification of reactive systems via instantiation of Parameterised Boolean Equation Systems. *Information and Computation*, 209(4):637–663, 2011, DOI: 10.1016/j.ic.2010.11.025.

[115] S. C. W. Ploeger, *Improved Verification Methods for Concurrent Systems*. Ph.D. thesis, Technische Universiteit Eindhoven, 2009, DOI: 10.6100/IR643995.

[116] A. Pnueli, The temporal logic of programs. In *SFCS 1977*, pp. 46–57, IEEE, 1977, DOI: 10.1109/SFCS.1977.32.

[117] J. van de Pol, Operations on Fixpoint Equation Systems, 2010, unpublished manuscript.

[118] J. van de Pol and M. Timmer, State Space Reduction of Linear Processes using Control Flow Reconstruction. In *ATVA 2009*, vol. 5799 of *LNCS*, pp. 54–68, 2009, DOI: 10.1007/978-3-642-04761-9_5.

[119] Y. S. Ramakrishna and S. A. Smolka, Partial-Order Reduction in the Weak Modal Mu-Calculus. In *CONCUR 1997*, vol. 1243 of *LNCS*, pp. 5–24, 1997, DOI: 10.1007/3-540-63141-0_2.

[120] M. Rodriguez, M. Piattini, and C. Ebert, Software Verification and Validation Technologies and Tools. *IEEE Software*, 36(2):13–24, 2019, DOI: 10.1109/MS.2018.2883354.

[121] K. Schmidt, Stubborn sets for model checking the EF/AG fragment of CTL. *Fundamenta Informaticae*, 43(1-4):331–341, 2000.

[122] S. F. Siegel, What's Wrong with On-the-Fly Partial Order Reduction. In *CAV 2019*, vol. 11562 of *LNCS*, pp. 478–495, 2019, DOI: 10.1007/978-3-030-25543-5_27.

[123] A. Tarski, A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955, DOI: 10.2140/pjm.1955.5.285.

[124] S. Tripakis and S. Yovine, Analysis of Timed Systems Using Time-Abstracting Bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001, DOI: 10.1023/A:1008734703554.

[125] Y. S. Usenko, *Linearization in μCRL*. Ph.D. thesis, Eindhoven University of Technology, 2002, DOI: 10.6100/IR560176.

[126] A. Valmari, A Stubborn Attack on State Explosion. In *CAV 1990*, vol. 531 of *LNCS*, pp. 156–165, 1991, DOI: 10.1007/BFb0023729.

[127] A. Valmari, Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, vol. 483, pp. 491–515, 1991, DOI: 10.1007/3-540-53863-1_36.

[128] A. Valmari, A Stubborn Attack on State Explosion. *Formal Methods in System Design*, 1(4):297–322, 1992, DOI: 10.1007/BF00709154.

[129] A. Valmari, The state explosion problem. In *ACPN 1996*, vol. 1491 of *LNCS*, pp. 429–528, 1996, DOI: 10.1007/3-540-65306-6_21.

[130] A. Valmari, Stubborn Set Methods for Process Algebras. In *POMIV 1996*, vol. 29 of *DIMACS*, pp. 213–231, 1997, DOI: 10.1090/dimacs/029/12.

[131] A. Valmari, Stop It, and Be Stubborn! *ACM Transactions on Embedded Computing Systems*, 16(2):46:1–46:26, 2017, DOI: 10.1145/3012279.

[132] A. Valmari and H. Hansen, Stubborn Set Intuition Explained. In *ToPNoC XII*, vol. 10470 of *LNCS*, pp. 140–165, 2017, DOI: 10.1007/978-3-662-55862-1_7.

[133] K. Varpaaniemi, On Stubborn Sets in the Verification of Linear Time Temporal Properties. *Formal Methods in System Design*, 26(1):45–67, 2005, DOI: 10.1007/s10703-005-4594-y.

[134] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub. In *ESEC/FSE 2015*, pp. 805–816, 2015, DOI: 10.1145/2786805.2786850.

[135] W. Wesselink, T. Neele, and T. A. C. Willemse, *PBES rewriters*. mCRL2 developer documentation, accessed 20-04-2020.

[136] W. Wesselink and T. A. C. Willemse, Evidence Extraction from Parameterised Boolean Equation Systems. In *ARQNL 2018*, vol. 2095 of *CEUR Workshop Proceedings*, pp. 86–100, 2018.

[137] T. A. C. Willemse, Consistent Correlations for Parameterised Boolean Equation Systems with Applications in Correctness Proofs for Manipulations. In *CONCUR 2010*, vol. 6269 of *LNCS*, pp. 584–598, 2010, DOI: 10.1007/978-3-642-15375-4_40.

[138] T. A. C. Willemse, Verification using Parameterised Boolean Equation Systems. *NVTI Nieuwsbrief*, pp. 51–58, 2010, URL `http://www.nvti.nl/Newsletter/Nieuwsbrief2010.pdf`.

[139] K. Wolf, Petri Net Model Checking with LoLA 2. In *Petri Nets 2018*, vol. 10877 of *LNCS*, pp. 351–362, 2018, DOI: 10.1007/978-3-319-91268-4_18.

[140] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998, DOI: 10.1016/S0304-3975(98)00009-7.

Summary

Reductions for Parity Games and Model Checking

The design and implementation of software systems has long been recognised to be a difficult task. In the field of theoretical computer science, formal methods aim to assist the design process by providing mathematical theories that allow one to reason about the behaviour of (concurrent) processes. One of the most effective techniques for analysing concurrent behaviour is model checking, which has seen many applications over the past 40 years. A typical model checking algorithm takes a high-level specification of a system's behaviour and a requirement in the shape of a formal property and checks whether the transition system underlying the specification satisfies the property. A fundamental challenge in model checking is the so-called state-space explosion: the arbitrary interleaving of the behaviour of many parallel processes causes an exponential blow-up in the size of the underlying transition system.

This thesis explores several new techniques that aim to address the state-space explosion problem by reducing parity games and their high-level encoding, parameterised Boolean equations systems (PBESs). A PBES is a sequence of Boolean equations augmented with data, where each equation is furthermore labelled with a fixpoint, allowing one to distinguish the largest and smallest solutions. PBESs can encode many types of decision problems, among them model checking problems. Computing the solution to a PBES answers the decision problem it encodes. PBES solving often involves generating its underlying parity game, which frequently are very large due to the state-space explosion, motivating the application of reduction techniques. All of the reductions we propose exploit the fact that PBESs allows very coarse abstractions, while (partially) preserving their solution. The contents of the thesis can be divided up into three parts.

The first part discusses symbolic techniques that enable solving of PBESs with an infinite underlying parity game. As a prerequisite, we introduce a normal form for PBESs that enables symbolic reasoning about the transitions it induces in the underlying game. The solving procedures proposed here perform partition

refinement, based on a bisimulation relation between the nodes of the parity game. This technique is provably more general than similar techniques for behavioural specifications. Furthermore, experimental results show that it also outperforms other tools. Our tool is the first implementation of a semi-decision procedure for deciding bisimilarity of arbitrary infinite processes.

The second part of the thesis discusses partial-order reduction (POR), which is a technique that aims to deal with the state-space explosion problem by not exploring all similar interleavings of parallel processes. First, we identify a theoretical problem in related work and show that one variant of POR is not guaranteed to preserve linear-time properties, contrary to what is claimed in an often-cited theorem. We propose a solution and show exactly in which derivative works the problem manifests itself. Subsequently, we apply partial-order reduction to PBESs and parity games. Compared to traditional POR approaches that operate on transition systems, this has the benefit that POR can also be applied when checking stutter-sensitive properties. Furthermore, the property is automatically taken into consideration during the reduction. Experiments show that substantial reductions can be achieved.

The final part deals with PBESs that contain quantifiers over data. Quantifiers that have a larger-than-necessary scope can cause the underlying parity game to grow unnecessarily. We introduce several techniques for transforming PBESs with quantifiers and show how their scope can be reduced. Furthermore, we discuss guards for PBESs: expressions that characterise when two equations depend on each other. Guards are a fundamental tool in several other static analysis techniques. The guards we compute are stronger than those proposed before, and thus contain more information on dependencies.

Samenvatting

Reducties voor Pariteitsspellen en Modelchecken

Al sinds de begindagen van computers wordt het ontwerpen en implementeren van softwaresystemen gezien als een complexe aangelegenheid. Binnen het onderzoeksveld van theoretische informatica zijn formele methoden ontwikkeld die het ontwerpproces ondersteunen door middel van theorie waarmee men over het gedrag van (parallelle) processen kan redeneren. Eén van de meest effectieve technieken voor het analyseren van parallel gedrag is modelchecken, dat veelvuldig is toegepast in de afgelopen 40 jaar. Een typisch modelcheckalgoritme heeft als invoer een abstracte specificatie van het gedrag van een systeem en een vereiste in de vorm van een formele eigenschap en controleert of het transitiesysteem dat gemodelleerd wordt door de specificatie, voldoet aan de eigenschap. De fundamentele uitdaging in modelchecken is de zogenaamde toestandsruimteontploffing: het arbitrair verweven van het gedrag van vele parallele processen leidt tot een exponentiële toename van de grootte van het onderliggende transitiesysteem.

Dit proefschrift verkent een aantal nieuwe technieken die het probleem van toestandsruimteontploffing pogen op te lossen door het reduceren van pariteitsspellen en hun compacte beschrijving, geparametriseerde booleaanse vergelijkingsstelsels (PBES). Een PBES is een sequentie van booleaanse vergelijkingen die zijn aangevuld met data en fixpunten, wat het mogelijk maakt om kleinste en grootste oplossingen te onderscheiden. PBESsen kunnen vele verschillende soorten beslisproblemen beschrijven, waaronder modelcheckproblemen. Het berekenen van de oplossing van een PBES beantwoordt ook het beslisprobleem dat erin beschreven wordt. Het oplossen van een PBES omvat vaak het genereren van het onderliggende pariteitsspel, dat regelmatig erg groot wordt door de toestandsruimteontploffing. Dit motiveert het toepassen van reductietechnieken. Alle reducties die we voorstellen maken gebruik van het feit dat PBESsen en pariteitsspellen erg grove reducties toestaan, terwijl hun oplossing (deels) behouden blijft.

De inhoud van dit proefschrift kan opgedeeld worden in drie delen. Het eerste deel bespreekt symbolische technieken die het mogelijk maken om PBESsen met een

oneindig onderliggend pariteitsspel op te lossen. Hiervoor is een normaalvorm voor PBESsen benodigd, die we eerst introduceren. De normaalvorm stelt ons in staat om symbolisch te redeneren over de transities die door de PBES geïnduceerd worden in het onderliggende spel. De oplosprocedures die hier voorgesteld worden passen partitieverfijning toe, gebaseerd op een bisimulatierelatie tussen de knopen van het pariteitsspel. Deze techniek is bewijsbaar generieker dan vergelijkbare technieken voor specificaties die gedrag beschrijven. Bovendien laten experimentele resultaten zien dat onze techniek ook in de praktijk beter presteert dan andere. Onze implementatie is de eerste implementatie van een semi-beslisprocedure voor het beslissen van bisimulariteit van willekeurige oneindige processen.

Het tweede deel van het proefschrift bespreekt partiële-ordereductie (POR), een techniek die het probleem van toestandsruimteontploffing probeert op te lossen door niet alle verwevingen van parallele processen te verkennen. Allereerst identificeren we een theoretisch probleem in verwant werk en tonen we aan dat een bepaalde variant van POR niet noodzakelijkerwijs lineaire-tijdeigenschappen behoudt, in tegenstelling tot wat beweerd wordt in een vaak geciteerde stelling. We stellen een oplossing voor en we laten zien in welke afgeleide werken het probleem zich voordoet. Vervolgens passen we partiële-orderreductie toe op PBESsen en pariteitsspellen. Vergeleken met traditionele POR-technieken die opereren op transitiesystemen, heeft onze aanpak als voordeel dat POR ook toegepast kan worden bij het checken van stottergevoelige eigenschappen en dat de eigenschap automatisch ook beschouwd wordt tijdens de reductie. Experimenten tonen aan dat significante reductie behaald kunnen worden.

Het laatste deel behandelt PBESsen die kwantoren over data bevatten. Kwantoren die een groter dan noodzakelijk deel van de PBES omvatten kunnen er toe leiden dat het onderliggende pariteitsspel onnodig groeit. We introduceren verschillende technieken voor het transformeren van PBESsen met kwantoren en laten zien hoe hun reikwijdte verkleind kan worden. Verder bespreken we afhankelijkheidscondities voor PBESsen. Deze karakteriseren de afhankelijkheid tussen twee vergelijkingen in een PBES, en zijn daarmee een fundamenteel gereedschap voor andere statische-analysetechnieken. De afhankelijkheidscondities die wij berekenen zijn sterker dan de condities die eerder zijn voorgesteld, en bevatten dus meer informatie.

Curriculum Vitae

Thomas Neele was born in Bilthoven and later moved to Doetinchem, where he grew up. After finishing high school, he started the Computer Science bachelor program at the University of Twente in Enschede. He obtained his BSc degree cum laude, and continued with the Computer Science master program, specialisation "Methods and Tools for Verification", at the same university. The research for his master's thesis was carried out at the Eindhoven University of Technology, under the supervision of dr.ing. Anton Wijs. The master's thesis titled "GPU Implementation of Partial-Order Reduction" was awarded the highest grade and later received the Ngi-NGN Informatie Scriptieprijs, second place, from Koninklijke Hollandsche Maatschappij der Wetenschappen. Thomas obtained his MSc degree cum laude.

Shortly after, Thomas started as a PhD candidate at the Eindhoven University of Technology in the group Formal Systems Analysis. His project in the IMPULS II program was partly funded by ASML and supervised by prof.dr.ir. Jan Friso Groote and dr.ir. Tim Willemse. His research focussed on improving the scalability of model checking techniques by applying reductions to parameterised Boolean equation systems and parity games. During this time, Thomas also played an active role in the mCRL2 development team. For his scientific contributions, Thomas received two best paper awards. His dissertation is titled "Reductions for Parity Games and Model Checking".

After his PhD, Thomas will start as a Post Doctoral Research Assistant at the Royal Holloway University of London, where he will investigate efficient model learning techniques for concurrent hardware.

Titles in the IPA Dissertation Series since 2017

M.J. Steindorfer. *Efficient Immutable Collections*. Faculty of Science, UvA. 2017-01

W. Ahmad. *Green Computing: Efficient Energy Management of Multiprocessor Streaming Applications via Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-02

D. Guck. *Reliable Systems – Fault tree analysis via Markov reward automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2017-03

H.L. Salunkhe. *Modeling and Buffer Analysis of Real-time Streaming Radio Applications Scheduled on Heterogeneous Multiprocessors*. Faculty of Mathematics and Computer Science, TU/e. 2017-04

A. Krasnova. *Smart invaders of private matters: Privacy of communication on the Internet and in the Internet of Things (IoT)*. Faculty of Science, Mathematics and Computer Science, RU. 2017-05

A.D. Mehrabi. *Data Structures for Analyzing Geometric Data*. Faculty of Mathematics and Computer Science, TU/e. 2017-06

D. Landman. *Reverse Engineering Source Code: Empirical Studies of Limitations and Opportunities*. Faculty of Science, UvA. 2017-07

W. Lueks. *Security and Privacy via Cryptography – Having your cake and eating it too*. Faculty of Science, Mathematics and Computer Science, RU. 2017-08

A.M. Şutîi. *Modularity and Reuse of Domain-Specific Languages: an exploration with MetaMod*. Faculty of Mathematics and Computer Science, TU/e. 2017-09

U. Tikhonova. *Engineering the Dynamic Semantics of Domain Specific Languages*. Faculty of Mathematics and Computer Science, TU/e. 2017-10

Q.W. Bouts. *Geographic Graph Construction and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2017-11

A. Amighi. *Specification and Verification of Synchronisation Classes in Java: A Practical Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-01

S. Darabi. *Verification of Program Parallelization*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-02

J.R. Salamanca Tellez. *Coequations and Eilenberg-type Correspondences*. Faculty of Science, Mathematics and Computer Science, RU. 2018-03

P. Fiterău-Broştean. *Active Model Learning for the Analysis of Network Protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2018-04

D. Zhang. *From Concurrent State Machines to Reliable Multi-threaded Java Code*. Faculty of Mathematics and Computer Science, TU/e. 2018-05

H. Basold. *Mixed Inductive-Coinductive Reasoning Types, Programs and Logic*. Faculty of Science, Mathematics and Computer Science, RU. 2018-06

A. Lele. *Response Modeling: Model Refinements for Timing Analysis of Runtime Scheduling in Real-time Streaming Systems*. Faculty of Mathematics and Computer Science, TU/e. 2018-07

N. Bezirgiannis. *Abstract Behavioral Specification: unifying modeling and programming*. Faculty of Mathematics and Natural Sciences, UL. 2018-08

M.P. Konzack. *Trajectory Analysis: Bridging Algorithms and Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2018-09

E.J.J. Ruijters. *Zen and the art of railway maintenance: Analysis and optimization of maintenance via fault trees and statistical model checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-10

F. Yang. *A Theory of Executability: with a Focus on the Expressivity of Process Calculi*. Faculty of Mathematics and Computer Science, TU/e. 2018-11

L. Swartjes. *Model-based design of baggage handling systems*. Faculty of Mechanical Engineering, TU/e. 2018-12

T.A.E. Ophelders. *Continuous Similarity Measures for Curves and Surfaces*. Faculty of Mathematics and Computer Science, TU/e. 2018-13

M. Talebi. *Scalable Performance Analysis of Wireless Sensor Network*. Faculty of Mathematics and Computer Science, TU/e. 2018-14

R. Kumar. *Truth or Dare: Quantitative security analysis using attack trees*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-15

M.M. Beller. *An Empirical Evaluation of Feedback-Driven Software Development*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2018-16

M. Mehr. *Faster Algorithms for Geometric Clustering and Competitive Facility-Location Problems*. Faculty of Mathematics and Computer Science, TU/e. 2018-17

M. Alizadeh. *Auditing of User Behavior: Identification, Analysis and Understanding of Deviations*. Faculty of Mathematics and Computer Science, TU/e. 2018-18

P.A. Inostroza Valdera. *Structuring Languages as Object-Oriented Libraries*. Faculty of Science, UvA. 2018-19

M. Gerhold. *Choice and Chance - Model-Based Testing of Stochastic Behaviour*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2018-20

A. Serrano Mena. *Type Error Customization for Embedded Domain-Specific Languages*. Faculty of Science, UU. 2018-21

S.M.J. de Putter. *Verification of Concurrent Systems in a Model-Driven Engineering Workflow*. Faculty of Mathematics and Computer Science, TU/e. 2019-01

S.M. Thaler. *Automation for Information Security using Machine Learning*. Faculty of Mathematics and Computer Science, TU/e. 2019-02

Ö. Babur. *Model Analytics and Management*. Faculty of Mathematics and Computer Science, TU/e. 2019-03

A. Afroozeh and A. Izmaylova. *Practical General Top-down Parsers.* Faculty of Science, UvA. 2019-04

S. Kisfaludi-Bak. *ETH-Tight Algorithms for Geometric Network Problems.* Faculty of Mathematics and Computer Science, TU/e. 2019-05

J. Moerman. *Nominal Techniques and Black Box Testing for Automata Learning.* Faculty of Science, Mathematics and Computer Science, RU. 2019-06

V. Bloemen. *Strong Connectivity and Shortest Paths for Checking Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-07

T.H.A. Castermans. *Algorithms for Visualization in Digital Humanities.* Faculty of Mathematics and Computer Science, TU/e. 2019-08

W.M. Sonke. *Algorithms for River Network Analysis.* Faculty of Mathematics and Computer Science, TU/e. 2019-09

J.J.G. Meijer. *Efficient Learning and Analysis of System Behavior.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-10

P.R. Griffioen. *A Unit-Aware Matrix Language and its Application in Control and Auditing.* Faculty of Science, UvA. 2019-11

A.A. Sawant. *The impact of API evolution on API consumers and how this can be affected by API producers and language designers.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2019-12

W.H.M. Oortwijn. *Deductive Techniques for Model-Based Concurrency Verification.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2019-13

M.A. Cano Grijalba. *Session-Based Concurrency: Between Operational and Declarative Views.* Faculty of Science and Engineering, RUG. 2020-01

T.C. Nägele. *CoHLA: Rapid Cosimulation Construction.* Faculty of Science, Mathematics and Computer Science, RU. 2020-02

R.A. van Rozen. *Languages of Games and Play: Automating Game Design & Enabling Live Programming.* Faculty of Science, UvA. 2020-03

B. Changizi. *Constraint-Based Analysis of Business Process Models.* Faculty of Mathematics and Natural Sciences, UL. 2020-04

N. Naus. *Assisting End Users in Workflow Systems.* Faculty of Science, UU. 2020-05

J.J.H.M. Wulms. *Stability of Geometric Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2020-06

T.S. Neele. *Reductions for Parity Games and Model Checking.* Faculty of Mathematics and Computer Science, TU/e. 2020-07