# Advanced Reduction Techniques for Model Checking

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 17 september 2013 om 14.00 uur

door

Jeroen Johan Anna Keiren

geboren te Venray

Dit proefschrift is goedgekeurd door de promotiecommissie:

| | |
|---|---|
| voorzitter: | prof.dr. E.H.L. Aarts |
| 1$^{e}$ promotor: | prof.dr.ir. J.F. Groote |
| copromotor(en): | dr.ir. T.A.C. Willemse |
| leden: | prof.dr.ir. T. Basten |
| | prof.dr. Y. Venema (UvA) |
| | prof.dr. M. Lange (Universität Kassel) |
| | prof.dr. F.W. Vaandrager (RUN) |
| | prof.dr.ir. W.M.P. van der Aalst |

# Advanced Reduction Techniques for Model Checking

Jeroen J. A. Keiren

IPA Dissertation Series 2013-11
ISBN: 978-90-386-3427-2
*A catalogue record is available from the Eindhoven University of Technology Library*

Accompanying data used in Chapter 5 is available via the following DOI:
doi:10.4121/uuid:ff67b0cb-91e0-4c14-a622-73219b9a8fc2

Typeset with LaTeX (TeXLive 2012)
Cover design by Kiek Ontwerp Studio
Printed by Printservice Eindhoven University of Technology, The Netherlands


TU/e Technische Universiteit **Eindhoven** University of Technology

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author was employed at the Eindhoven University of Technology.

# Contents

# Preface

I vividly recall the moment, somewhere in February 2009, when Jan Friso Groote walked into my office. We had discussed my ambition for doing scientific research before, and he offered me a position in his group. I was still finishing my MSc thesis, but, provided that I finished my MSc, I could start my PhD in September. I would like to express my gratitude to Jan Friso Groote for allowing me to pursue my PhD. Jan Friso also gave me the opportunity to freely explore my own fields of interest, and go the direction that I saw fit at the time.

I would also like to thank the committee for agreeing to review my thesis, and serve as an opponent during the defence: Wil van der Aalst, Emile Aarts, Twan Basten, Martin Lange, Frits Vaandrager and Yde Venema.

Ultimately, I believe it was Michel Reniers who provided the spark that lit my scientific interest. Back in 2006 there was a position for a student assistant in the group that was then called *Ontwerp en Analyse van Systemen*. The candidate was expected to extend and improve tools in the mCRL2 toolset. Michel pointed out to Jan Friso that I might be interested in this job. In the end, I started working on mCRL2, together with Carst Tankink, in the summer of 2006. While doing this work it became more and more clear that I was interested in improving the way in which we develop software. I am confident that this, at least partly, was due to my involvement in the development of formal methods tools. Eventually, this interest in formal methods resulted in a graduation project that I carried out under supervision of Tim Willemse in 2009.

At the MSc diploma ceremony, Tim said he wished for a long and fruitful collaboration in the future. I wonder whether he will be more careful with such phrasings in the future, since he definitely got what he wished for. Tim, thank you for supervising me in the way that you did. I would like to thank you for the great many things you taught me about doing research, and for asking the right questions at the right time. Looking at the papers that we have published together, I think our collaboration was, indeed, fruitful, and I hope there might be more to come.

I would also like to take a moment to thank some people in particular, although I apologise up-front, since I will undoubtedly forget someone. I thank my co-authors for their collaboration in the papers that ended up in this thesis. Sjoerd Cranen, Wieger Wesselink, Tim Willemse and Michel Reniers, thank you for inspiring, sometimes long, discussions on the topic of this thesis. Related to the topic of this thesis, the work on mCRL2 resulted in publications together with the other developers. Jan Friso Groote, Aad Mathijssen, Bas Ploeger, Frank Stappers, Carst Tankink, Yaroslav Usenko, Erik de Vink, Muck van Weerdenburg, Wieger Wesselink, Tim Willemse, and Jeroen van der

Wulp, thank you for including me in a project in which I could develop my software engineering and project management skills. Finally, I would like to thank the co-authors with whom I carried out industrial case studies. Martijn Klabbers, Yi-Ling Hwong, Vincent Kusters and Sander Leemans, thank you for your collaboration. I should also thank the CMS group at CERN, especially Frank Glege, for allowing me to work on verifying the finite state machines of the detector control system, and allowing me to visit CERN.

During the past four years at the design and analysis of systems group which is now called Formal System Analysis, and has become part of the section Model Driven Software Engineering, I have worked with a large number of interesting people. First of all, I would like to thank my office mates, in chronological order Jeroen van der Wulp, Wieger Wesselink, Tim Willemse, Maciej Gazda, Harsh Beohar and Neda Noroozi. Of course, a word of thanks goes to all other people whom I have had the pleasure to work with over the years. I would like to give a special word of thanks to Tineke van den Bosch, the group secretary, for always being there for a nice chat, and for her advice in organisational matters.

My time as a PhD student would not be complete without the support from the research school, IPA, and the events they organised. Meivan Cheng, Michel Reniers and Tim Willemse, thank you for organising these events. Also a big thanks to the friends I met at the IPA days. Especially thanks to the IPA regulars Alexandra Silva, Carst Tankink, Cynthia Kop, Gijs Kant, José Pedro Magalhães, Mark Timmer, Paul van Tilburg, Pim Vullers, and Stephanie Kemper, who made the IPA days into memorable events.

Closer to home, I would like to thank my parents, Marcel and Ria, and my sisters Suzanne and Mariëlle for supporting me in the choices that I have made, and for being interested in the research that I do. I know you probably did not get most of what I told about my research, yet you still showed an interest in what I do. A word of thanks also to Malou's parents, Toos and René, for putting up with me during the past decade.

Finally, I would like to thank Malou, my love, for her patience, especially during the past months. Sorry for neglecting you while finishing my thesis. I will be happy to return the favour, once it is your turn to finish yours.

*Jeroen Keiren, April 2013*

# Chapter 1

# Introduction

Since their first appearance the use of computers has spread widely. From mainframes in the early days, computers evolved to desktop- and laptop computers, and more recently, tablet devices. The majority of software is, however, found as *embedded software* in devices ranging from small home appliances such as telephones, televisions and stereo sets, to safety critical systems such as the vehicle control systems in cars and airplanes, or systems controlling (access to) infrastructure such as bridges and tunnels. Complex software also occurs in large corporate systems, where decisions are made by these software systems autonomously.

The amount of embedded software increases by 10 to 20 percent per year [EJ09]. The increase in complexity of embedded software is, *e.g.*, illustrated by the on board software in cars. Today, cars can contain close to 100 million lines of code, and run on up to 100 different computation units [Cha09]. Currently, this software is still mainly concerned with support tasks and engine management, but manufacturers are also pioneering drive-by-wire technology [Bre01], in which *e.g.*, steering is controlled by a combination of electronics and embedded software connected to the steering wheel. In avionics, these technologies are already commonplace, and, for fighter aircraft, fly-by-wire techniques are necessary due to the inherent instability of the aircraft [Col99]. It is expected that in the upcoming years cars will increasingly use drive-by-wire technology.

The omnipresence of software, and the rapid increase in the amount of software, results in the challenge for computer science to develop techniques for constructing large dependable systems. Since the early days of computers, scientists have been looking for formal techniques to facilitate this. This has resulted in a wide variety of verification approaches, of which we discuss the history in the next section.

Some of the verification approaches we discuss have also been used to verify hardware designs. More recently, verification techniques have been employed to analyse biological models [Hea+08].

## 1.1 Historical Context

The need for *software verification* was observed as early as the 1940s by Alan Turing and John von Neumann [Tur49; MJ84; GN47] who proved correctness of small programs by hand. Consequently further proof-based techniques were developed, such as Floyd's assertional reasoning on flowcharts [Flo67], Hoare's axiomatic semantics for programming constructs [Hoa69], popularly known as *Hoare triples*, and Dijkstra's weakest precondition calculus [Dij75; Dij76]. In these early times, programs were typically viewed as input/output functions, *i.e.*, the program reads an input, performs a number of computation steps, and then produces an output. We still find such programs today in the form of command line applications. With this type of program we typically associate Turing machines [Tur37] as the standard model of computation. In this model of computation, we essentially view the program as a function on the input. Turing showed that every computable function can be computed using a Turing machine.

In the 1980s, the view of computer programs shifted from these input/output functions to *interactive processes*. Instead of a function that computes such an input/output relation, a computer program becomes a collection of concurrent processes, interacting with each other and with the environment. In this setting processes themselves can again be the result of composing other processes. Arguably, this view of programs is more natural in our environment, in which computers operate autonomously, and interact with the environment every now and then. Models of computation were established to reason about concurrent processes. Prominent examples of such models are Petri nets [Pet62], process algebras such as ACP [BK84], CSP [Hoa85] and CCS [Mil80], and the $\pi$-calculus [MPW92a; MPW92b]. In these models, *computations* and *interactions* of processes can be specified, such that their *behaviour* can be studied.

Methods have been developed to automatically establish correctness of concurrent processes using these models of computation. This process is typically referred to as *verification*. We can roughly distinguish three verification approaches: *theorem proving*, *equivalence checking* and *model checking*. All three of them are used to improve reliability of critical systems such as the on-board systems of cars, and avionics systems. Ultimately, these methodologies are employed to verify that undesired behaviour never occurs, *i.e.*, that a model of the system is *safe*, or that desired behaviour occurs eventually, also referred to as *liveness*. We next discuss the most important aspects of each of the approaches.

**Theorem proving.** In theorem proving, properties about programs—or models of these programs—are proven with the help of theorem provers. Theorem provers come in two categories, automated theorem provers and interactive theorem provers. *Automated theorem provers* [Har09], typically SAT solvers such as Chaff [Mos+01] or Minisat [ES04], and SMT solvers such as Yices [DM06] or Z3 [MB08], can prove properties without user interaction. These techniques are powerful for proving some classes of properties, but in general fail once the properties get more complicated. When automated theorem provers are insufficient for proving correctness of a system, one can resort to the use of *interactive theorem provers* [Rob01]—also referred to as *proof assistants*—such as PVS [ORS92], Coq [BC04], or Isabelle [Pau89]. These are generic tools that allow the user to prove mathematical properties in a rigorous way. The main disadvantage of these tools is that

proving properties requires a great effort from the user.

**Preorder- and Equivalence checking.** An alternative approach to establishing program correctness is *preorder- or equivalence checking* [CS01]. In this approach, a compact description—the specification—of the system under analysis is constructed. Due to its compact representation, correctness of the specification can be easily established. It is then checked whether a more detailed description—the implementation—supersedes the specification, in case of preorder checking, or is equivalent to the specification, in case of equivalence checking.

An equivalence or a preorder is a relation between a specification and an implementation. The specific relation that is used depends on the type of property that one wants to verify. In case we are only interested in linear-time properties, *i.e.*, properties that reason about *traces*, or *executions* of the program, *trace equivalence* or *trace inclusion* [Hoa80] can be used. If branching-time properties need to be considered, then typically *strong bisimulation* is used [Par81; Mil89]. In addition, some relations allow to abstract from behaviour that is considered to be *internal* to a process. Prominent examples of such relations are *weak traces* [BHR84], *branching bisimulation* [GW96] and *stuttering equivalence* [BCG88]. The choice for a relation is not restricted to the ones we mentioned so far. In fact, the choice is virtually unlimited, as was nicely illustrated by van Glabbeek in his seminal papers [Gla90; Gla93].

Some of the relations serve as the underlying semantics of process algebras, for example, *traces* and *failures* underlie CSP [Hoa85], and *strong bisimulation* underlies, *e.g.*, CCS [Mil80] and ACP [BK84]. Several tools, based on these process algebras, use preorder- and equivalence checking techniques. Some examples of such tools are FDR [For], CADP [Gar+11], the Concurrency workbench [CPS93] and mCRL2 [Cra+13].

**Model checking.** In model checking [BK08], a model is developed together with the properties that it should satisfy. The properties are typically expressed as formulae in a modal or temporal logic such as LTL [Pnu77], CTL [CE82], CTL$^*$ [EH86], or the $\mu$-calculus [Koz83]. The model is represented as a finite state machine, also called labelled transition system. The approach then relies on checking whether the model satisfies the property in a fully automated fashion. Model checking was pioneered by Emerson, Clarke and Sifakis [EC80; CE82; QS82; CES86], who were awarded the A.M. Turing award for their groundbreaking work on model checking in 2007 [ACM07].

The classical approach to model checking, which is still commonly used in the verification of embedded systems, is explicit-state model checking, in which state spaces are stored explicitly. The main drawback of this approach is that it suffers from the infamous *state space explosion problem*: the number of states in the system grows exponentially with the number of parallel components. For real-life systems it is typically not possible to store all states in memory.

A vast number of different approaches has been developed to counter the state space explosion problem. A popular technique, especially in the verification of hardware systems, is *symbolic model checking* [Bur+90], which uses BDDs [Bry86] for compactly storing state spaces. Popular tools using symbolic model checking are nuSMV [Cim+00] and Prism [KNP11], of which the latter mainly focusses on verification of probabilistic systems, including biological models.

Another approach, that is particularly effective for bug hunting [vGK99], but less adequate for showing the absence of bugs, is *bounded model checking* [Bie+99]. Here a description of the state space, up to a bounded depth, is combined with the property under investigation, and encoded as a satisfiability problem. The resulting problem is then solved using one of the popular satisfiability solvers. The strength of this sound approach is that it typically finds bugs quickly, even in extremely large systems. This, however, comes at the cost of losing completeness, *i.e.*, if a bug is hidden in the system beyond the depth for which is checked, the bug will not be uncovered.

In model checkers based on process algebra, techniques have been developed that allow *symbolic manipulation* of an abstract, process algebraic description of a model before generating the state space. These manipulations aim to preserve validity of entire classes of properties, *e.g.* all properties formulated in the modal $\mu$-calculus, and can lead to dramatic reductions in the size of the explicit state space that is eventually generated. These techniques have been implemented in, *e.g.* the tool sets mCRL2 [Cra+13] and CADP [Gar+11].

**Verification using intermediate representations.** In the search for more effective verification methodologies, alternative model checking and equivalence checking approaches have been developed in which intermediate representations such as *parity games* and *Boolean equation systems (BESs)* are used. Mapping multiple verification problems onto a single intermediate representation ensures that advances in solving the intermediate representation immediately result in advances in solving these different verification problems.

The main advantage of this approach is that all available information is translated into a single representation. The model checking problem, *e.g.*, is translated into an intermediate representation in polynomial time. The information of both the state space and the property that is being checked is combined into a single representation. As a result of this translation, information that is not needed for deciding the outcome of the verification problem is automatically removed from the result. As a consequence, solving the resulting parity game or equation system is potentially more effective than solving the original verification problem directly.

A parity game [EJ91; McN93; Zie98; GTW02] is a two-player game played on a finite directed graph, which, in general, is assumed to be total. Every vertex in the graph is owned by one of two players, typically referred to as *even* and *odd*, and every vertex gets assigned a non-negative integer priority. The game is played by placing a token on some initial vertex, and moves are made according to the following simple rule. If the token is on a vertex owned by player *even*, then she moves the token along one of the outgoing edges of the vertex, otherwise *odd* decides the move. Since the graph is assumed to be total, this game can be played indefinitely. We refer to an infinite path in this game as a play. The play is won by player *even* if the parity of the least priority that occurs infinitely often in the play is even, otherwise it is won by player *odd*. If player *even* has a strategy with which she can win every play that starts in a given vertex, regardless of the moves made by her opponent, we say that this vertex is won by player *even*. It is well-known that parity games are determined, *i.e.*, every vertex is won by exactly one of the two players, and that for this it is sufficient to consider memoryless strategies, *i.e.*, every time the token hits a vertex, the player that owns the vertex makes the same choice

[Mar75; GH82; EJ91; McN93]. The problem of solving parity games, *i.e.* deciding for every vertex by which player it is won, is known to be in NP∩co-NP. A lot of authors have investigated the question whether a polynomial time algorithm exists [Jur00; VJ00; BV01; PV01; BSV03; Lan05; JPZ06; Obd06a; Sch07; GS08; Sch08a; CGR12; FL12]. It has been long thought that some of these algorithms might be polynomial, however, Oliver Friedmann has provided classes of examples for which these algorithms indeed require an exponential running time [Fri09; Fri10; Fri13; Fri11c; Fri11a; FHZ11; Fri11b], hence the question whether parity games can be solved polynomially is still open. Parity games and their background are introduced in detail in Section 2.6.

Instead of explicitly constructing the arena, like in the translation to parity games, equivalence checking and model checking can also be phrased as a two-player game [Sti97]. For model checking, *e.g.*, the game proceeds by simultaneously traversing states of the system and subformulae. Rules and winning criteria are similar to those in parity games.

Parity games are known to be linear time equivalent to Boolean equation systems [Lar93; Mad97]. These are systems of fixed point equations over the Boolean lattice. Boolean equation systems allow for a more compact representation of model checking problems than parity games by mixing Boolean connectives in the right hand sides of equations—in some cases there is no need for introducing intermediate equations, whereas in parity games intermediate vertices are needed. However, every Boolean equation system can be transformed to an equation system in which each right hand side contains only conjunctions or only disjunctions in polynomial time. If then also occurrences of the Boolean constants are removed, there is a direct one-to-one correspondence with parity games. Note that the only reason to remove the Boolean constants is to satisfy the totality requirement for the resulting parity game. Boolean equation systems are introduced in detail in Section 2.5; in Section 2.6 we detail the correspondence with parity games.

*Parameterised Boolean equation systems (PBESs)* generalise Boolean equation systems by parameterising equations with variables, and allowing universal and existential quantifications in right hand sides. This allows for more compact descriptions of a verification problem—part of the problem is encoded in the data. Furthermore, verification problems of symbolic process descriptions can be translated to parameterised Boolean equation systems [GW05a; GW05b].

## 1.2 Reduction of (Parameterised) Boolean Equation Systems and Parity Games

Like state spaces, (parameterised) Boolean equation systems and parity games happen to suffer from the state space explosion problem. This means that in practice they become too large to be effectively solvable. In order to deal with this, it seems natural to study reduction techniques for PBESs, BESs and parity games that counter this state space explosion problem. Such reduction techniques are studied in this thesis.

For labelled transitions systems and symbolic process descriptions equivalences and reduction techniques have been widely studied, see *e.g.*, [Par81; Gla90; Gla93; GL02]. However, for the intermediate representations that we consider in this thesis, reduc-

tion techniques have hardly been investigated. For parity games heuristics have been described that can be used to reduce the games, see, *e.g.*, [FL09]. Furthermore, the notions of direct- and delayed simulation were studied [FW06]. For parameterised Boolean equation systems basic static analysis techniques were described in [OWW09].

In Chapters 3 and 4 we combine the intermediate representations, in particular parity games and Boolean equation systems, with reduction techniques inspired by strong bisimulation [Par81] and branching bisimulation [Gla93] of labelled transition systems. Using this approach, we address the state space explosion apparent in Boolean equation systems and parity games. The potential of this method is established experimentally in Chapter 5.

The drawback of this approach is that it requires the explicit construction of the equation system or parity game before actually doing the reduction. Ultimately, we are looking for ways in which we can perform an *a priori* reduction of parameterised Boolean equation systems, such that the state space explosion is avoided altogether. As a step towards this goal we investigate static analysis techniques that are able to reduce parameterised Boolean equation systems symbolically in Chapter 6. Here we improve upon existing symbolic reduction techniques that were studied in [OWW09], and we rely on a generalisation of the equivalences from Chapter 3, due to Willemse [Wil10], for proving the correctness of our static analysis technique.

We are confident that the state space explosion problem can be addressed even more effectively by symbolic reductions based on the weaker equivalence notions that we describe in Chapter 4. Essentially, the equivalences in that chapter are inspired by branching-bisimulation equivalence of labelled transition systems. As a consequence, these equivalences essentially serve as a semantical basis for reduction techniques for PBESs that are inspired by, *e.g.*, cones and foci [GS01; FP03; FPP06] and confluence reduction [GS95; GS96; GP00; BP02].

Next we discuss the contributions made in this thesis in more detail. From a complexity-theoretic point of view, Boolean equation systems and parity games are equivalent. However, it is unclear how the transformation from Boolean equation systems to parity games affects the effectiveness of heuristics. Towards understanding this, in Chapter 3 we investigate the following question.

> Can the structure of arbitrary Boolean equation systems be systematically captured by a graph?

This question is answered positively by introducing the notion of structure graphs. Inspired by the equivalence checking approach to model checking, we propose strong bisimulation reduction of structure graphs as a technique to reduce the size of the underlying equation systems. The effect of the translation of Boolean equation systems to parity games—later referred to as normalisation—is then investigated in the context of strong bisimulation reduction.

> Does normalisation of a structure graph affect the reducing capabilities of strong bisimulation?

It turns out that normalisation does not negatively affect the reduction that can be achieved using strong bisimulation, and, in fact, we show that there are examples in

which normalisation results in a reduction by an arbitrarily larger factor than in the original equation system.

We generalise the weaker notion of *idempotence identifying bisimilarity* that was proposed in [KW11] to structure graphs. We investigate whether, also on structure graphs, this is indeed capable of identifying vertices representing idempotent formulae. So, are vertices representing $f \wedge f$ and $f$ equivalent modulo idempotence identifying bisimilarity? We answer this question negatively for the general case, and we show that idempotence is only identified in structure graphs corresponding to the subset of equation systems in simple recursive form.

Another question that beckons answering once a notion of strong bisimilarity for structure graphs is established is the following.

> Do bisimilar states in the state space give rise to bisimilar states in the Boolean equation systems encoding model checking problems?

To answer this question, we fix a translation of the model checking problem to Boolean equation systems. Given this translation, we prove the even stronger result that given a model checking problem, bisimilar states in state spaces that have been abstracted with abstractions safe with respect to the model checking problem lead to bisimilar states in the Boolean equation systems encoding the model checking problem.

Since structure graphs come with quite a heavy notational overhead, and, at least for the bisimulation setting, normalising a Boolean equation system before reduction using strong bisimulation can be beneficial, in Chapter 4 we turn our attention to parity games. In this chapter we investigate two main questions. The first one is the following.

> Can stuttering equivalence be used to reduce parity games?

We answer this question positively, and show that reduction modulo stuttering equivalence is sound. Since our notion of stuttering equivalence only allows vertices to be related if they are owned by the same player and have the same priority, vertices with a single outgoing edge, but owned by different players, may not be related. Intuitively, if a vertex in a parity game has a single outgoing edge, there is no real choice to be made in that vertex, hence the player is irrelevant.

> Can stuttering equivalence on parity games be weakened to take forced moves into account?

This second question leads to the notion of *governed stuttering equivalence*. We develop an algorithm for deciding governed stuttering equivalence. Also, we describe strong bisimulation and *governed bisimulation*—which is the parity game equivalent of idempotence identifying bisimulation, on parity games, and we present the lattice of equivalences that this results in.

The development of this lattice inevitably raises the following question.

> How effective are the equivalences that we introduce in reducing the size of parity games, and in reducing the time required for model checking?

These questions are investigated in Chapter 5. However, since there are no standard benchmarks for parity game solvers, and heuristics for reducing parity games, we start

Chapter 5 by proposing a standard benchmark suite that subsumes the set of parity games that have been used for experiments in the literature.

The drawback of using equivalence reductions on parity games—or structure graphs for that matter—is that the approach still requires the graph to be constructed explicitly. In Chapter 6 we study parameterised Boolean equation systems, which essentially are symbolic descriptions of, possibly infinite, Boolean equation systems, *i.e.*, they are fixed point equation systems where the variables are parameterised with data. Some static analysis techniques for reducing parameterised Boolean equation systems were described in [OWW09]. In this thesis, we study the following question.

> Can the static analysis techniques for parameterised Boolean equation systems be improved by taking the control flow into account?

We define a notion of control flow for PBESs, and on top of this we build such a static analysis technique. In proving the correctness of our static analysis we use a generalisation of the bisimulation results of Chapter 3. We evaluate the static analysis using the part of the benchmarks in Chapter 5 in which parameterised Boolean equation systems are used as an intermediate format in the parity game generation process.

To summarise, the results that are presented in Chapters 3 and 4 provide a semantical, coinductive basis for techniques for reducing parity games. In Chapter 5, we show that indeed, significant reductions in the sizes of parity games are possible. In Chapter 6 we show that reductions can also be achieved by a syntactic analysis of parameterised Boolean equation systems, while maintaining equivalence of the underlying parity games.

The work in this thesis contributes to the understanding of the inherent complexity of parity games and Boolean equation systems. The final chapter shows that the equivalences are suitable for the development of static analysis techniques of parameterised Boolean equation systems. The weak equivalences described in Chapter 4 open up the way for the development of more advanced symbolic reduction techniques for PBESs, improving our ability to combat the state space explosion problem in the verification of embedded software.

## 1.3 Origin of the Chapters

All chapters in this thesis have been written in such a way that they are mostly self-contained. We provide a common background for the techniques and formalisms that are used in Chapter 2. For each of the chapters we give their origin, and the relevant sections in Chapter 2 below.

### Chapter 3

The idea of using bisimilarity and idempotence identifying bisimilarity to reduce Boolean equation systems was first described in:

[KW11] J.J.A. Keiren and T.A.C. Willemse. Bisimulation Minimisations for Boolean Equation Systems. In HVC 2009. LNCS, Vol. 6405, pp. 102–116. Springer. 2011

Structure graphs generated from Boolean equation systems using structural operational semantics were described in:

[KRW12] J.J.A. Keiren, M.A. Reniers and T.A.C. Willemse. Structural Analysis of Boolean Equation Systems. In ACM TOCL 13(1): 8-1/35, 2012.

This chapter uses the latter paper as a basis, and extends the paper with the ideas on idempotence identifying bisimilarity from the first paper.

For this chapter it is recommended that Sections 2.2, 2.4 and 2.5 are read first.

## Chapter 4

This chapter combines the results from the following two papers:

[CKW11] S. Cranen, J.J.A. Keiren and T. A. C. Willemse. Stuttering Mostly Speeds Up Solving Parity Games. In NFM 2011, LNCS, Vol. 6617, pp. 207–221. Springer. 2011.

[CKW12b] S. Cranen, J.J.A. Keiren and T. A. C. Willemse, A Cure for Stuttering Parity Games. In ICTAC 2012, LNCS, Vol. 7521, pp.198–212. Springer. 2012.

Both papers are collaborative work with Sjoerd Cranen and Tim Willemse. In the first paper, the contributions of the author are the definition of stuttering bisimilarity, the proof that stuttering bisimilar vertices are won by the same player, and the experimental evaluation. This paper is, therefore, entirely contained in Chapter 4.

In the second paper, the author contributed to the definition of governed stuttering bisimilarity. The author's main effort in the paper resulted in the decidability results, on which we focus in this thesis. The remaining results, especially equivalence results of governed stuttering bisimilarity, and the proof that governed stuttering bisimilar vertices are won by the same player were contributed by Sjoerd Cranen; for these results, we refer to [CKW12b], and Sjoerd Cranen's forthcoming PhD thesis.

For this chapter it is recommended that Section 2.6 is read first.

## Chapter 5

Part of the benchmark sets that we describe in this chapter were used to evaluate the effect of stuttering equivalence and governed stuttering equivalence in [CKW11] and [CKW12b]. Some of the Boolean equation system examples were also used in [KW11]. Additional details are given in Appendix A.

The novel contribution of this chapter is the description of a benchmark set for parity games that subsumes the benchmarks used in the literature that we are aware of, and the implementation of a framework that can be used to generate these parity games and run experiments on them.

It is recommended that Sections 2.4, 2.5 and 2.6 are read first.

## Chapter 6

The static analysis technique—stategraph—that is presented in Chapter 6 was described in the following paper, that is currently under submission.

[KWW13] J.J.A. Keiren, J.W. Wesselink and T.A.C. Willemse. Improved Static Analysis of Parameterised Boolean Equation Systems using Control Flow Reconstruction. Submitted.

The experimental evaluation in this chapter is based on the relevant part of the benchmarks in Chapter 5.

This chapter is completely self-contained, apart from the fixed point theory of Section 2.3.

## 1.4 Other Contributions

This thesis focusses on the theoretical contributions to verification using parity games and parameterised Boolean equation systems. A number of contributions are not part of this thesis. These contributions mainly come in two categories: (1) practical results that are of importance to the implementation of model checking tools—especially mCRL2—or academic software development in general, and (2) applications of model checking to industrial cases.

The publications related to the implementation of model checkers are:

[Gro+11] J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, J.W. Wesselink and T.A.C. Willemse. Experiences in developing the mCRL2 toolset. In Software: Practice and Experience, 41(2): 143–153, 2011.

[Cra+13] S. Cranen, J.F. Groote, J.J.A. Keiren, F.P.M. Stappers, E.P. de Vink, J.W. Wesselink and T.A.C. Willemse. An Overview of the mCRL2 Toolset and its Recent Advances. In: TACAS 2013.

Applications of model checking are described in the following papers:

[KK12] J.J.A. Keiren, M.D. Klabbers. Modelling and verifying IEEE Std 11073-20601 session setup using mCRL2. Proc. 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012). In ECEASST, 2012.

[Hwo+13] Y.L. Hwong, J.J.A. Keiren, V.J.J. Kusters, S. Leemans, T.A.C. Willemse. Formalising and Analysing the Control Software of the Compact Muon Solenoid Experiment at the Large Hadron Collider. Accepted for publication in Science of Computer Programming.

# Chapter 2

# Model Checking

The work in this thesis is motivated by solving the model checking problem. Solving this problem means deciding whether a given behavioural specification satisfies a temporal or modal formula.

We describe behavioural specifications using labelled transition systems, which are introduced in Section 2.1. These can be compared and reduced using strong bisimulation, which is introduced in Section 2.2. We introduce basic fixed point theory in Section 2.3. This serves as a basis for the modal $\mu$-calculus, that we use to formulate model checking problems. The $\mu$-calculus is introduced in Section 2.4. The required fixed point theory is introduced in Section 2.3.

The work in this thesis is based on (parameterised) Boolean equation systems and parity games. These formalisms can be used for solving various verification problems such as the model checking problem and the equivalence checking problem. Boolean equation systems are introduced in Section 2.5, in which we also show how the model checking problem can be encoded as an equation system. In Section 2.6 we introduce parity games and their relation to Boolean equation systems. Parameterised Boolean equation systems generalise Boolean equation systems, by allowing abstract data types. They are only used in Chapter 6, and we therefore defer their introduction to that chapter.

## 2.1 Labelled Transition Systems

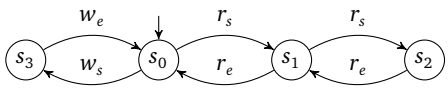
The behaviour of reactive systems is typically described using Kripke structures or labelled transition systems. In this thesis, we use labelled transitions systems (LTSs) to provide a formal, semantical model for the behaviour of a reactive system.

**Definition 2.1.** A *labelled transition system (LTS)* is a tuple $L = \langle S, s_0, Act, \rightarrow \rangle$, consisting of a non-empty set of states $S$, an initial state $s_0$, a non-empty set of actions $Act$ and a transition relation $\rightarrow \subseteq S \times Act \times S$.

For most of the theory in this thesis the initial state is irrelevant, hence we typically omit it. In Chapters 3–5 we assume that $S$ and $Act$ are finite—effectively this assumes that LTSs are finitely branching—whereas in Chapter 6 we also allow infinite-state systems in which also $Act$ is infinite.

We visualise labelled transition systems by directed, edge-labelled graphs. In line with this graphical notation, we write $s \xrightarrow{a} s'$ if and only if $(s, a, s') \in \rightarrow$.

**Example 2.2.** An LTS modelling mutual exclusion between two readers and a single writer is given below. The identity of the readers is immaterial to the validity of the mutual exclusion property, and is therefore excluded from the LTS.




Reading is started using an action $r_s$ and action $r_e$ indicates its termination. Likewise for writing. The states in this LTS are $\{s_0, s_1, s_2, s_3\}$.

## 2.2 Bisimulation

One common verification method is based on checking equivalence between LTSs. For equivalence checking there is a large number of equivalences that we can choose from, depending on the kind of properties that should be preserved by the equivalence, and whether we are interested in linear-time or branching-time properties. For linear-time properties, *trace equivalence* [Hoa80] is popular, for branching-time properties common choices are *strong bisimulation* [Par81] and *branching bisimulation* [GW96].

In this thesis we are mainly concerned with strong bisimulation.

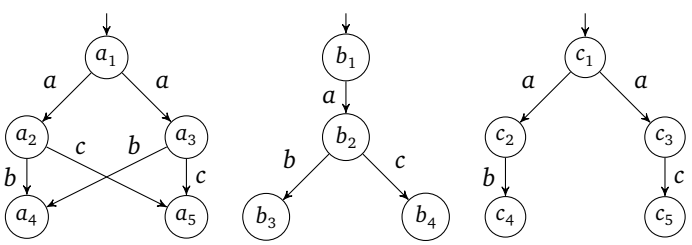
**Definition 2.3.** Let $L = \langle S, Act, \rightarrow \rangle$ be a labelled transition system. A symmetric relation $R \subseteq S \times S$ is a *strong bisimulation* if for all $(s, s') \in R$

$$\forall a \in Act, t \in S : \; s \xrightarrow{a} t \implies \exists t' \in S : \; s' \xrightarrow{a} t' \wedge (t, t') \in R$$

States $s, s' \in S$ are *bisimilar*, denoted $s \leftrightarroweq s'$, if and only if there is a bisimulation relation $R$ that relates states $s$ and $s'$.

We say that two labelled transition systems are bisimilar if their initial states are bisimilar.

**Example 2.4.** Consider the following three LTSs, in which the labels on the states are just for reference. The first two are bisimilar, witnessed by the relation relating $(a_1, b_1)$, $(a_2, b_2)$, $(a_3, b_2)$, $(a_4, b_3)$ and $(a_5, b_4)$. The second and the third are not related; for any bisimulation relation, $b_1$ and $c_1$ must be related, which in turn requires that $b_2$ is related to $c_2$ and $c_3$. However, the condition from the definition of bisimulation fails, since, *e.g.*, $c_2$ cannot mimic the $c$ action that can be done from $b_2$.



## 2.3 Lattices and Fixed Points

A large part of this work is based on the theory of fixed points, of which we first introduce the basics.

A binary relation $\leq$ on a set $S$ is a *partial order* if and only if it is reflexive, antisymmetric and transitive. A set $S$ equipped with a partial order $\leq$ is called a *partially ordered set* or *poset* $(S, \leq)$. Given a poset $(S, \leq)$, and $T \subseteq S$, the least- and greatest elements do not necessarily exist. If it exists, the *least element* of $T$ is $x \in T$ such that for all $y \in T$, $x \leq y$, and similarly the *greatest element* of $T$ is $x \in T$ such that for all $y \in T$, $y \leq x$.

For a poset $(S, \leq)$ and $T \subseteq S$, $x \in S$ is an *upper bound* of $T$ if and only if for all $y \in T$, $y \leq x$, similarly, $x \in S$ is a *lower bound* of $T$ if and only if for all $y \in T$, $x \leq y$. If the set of upper bounds of $S$ has a least element, we refer to it as the *least upper bound*, or *supremum*, denoted by $\bigvee S$. Similarly, if the set of lower bounds of $S$ has a greatest element, we refer to it as the *greatest lower bound*, or *infimum*, denoted by $\bigwedge S$.

**Definition 2.5.** A poset $(S, \leq)$ is called a *lattice* if and only if for every pair of elements $x, y \in S$ the least upper bound and the greatest lower bound exist. If $\bigvee T$ and $\bigwedge T$ exist for all subsets $T \subseteq S$, then $(S, \leq)$ is a *complete lattice*.

We denote the least element in $S$ in a complete lattice $(S, \leq)$ with $\bot$ and its greatest element with $\top$.

**Example 2.6.** In particular, given a set $S$, the power set of $S$, denoted $\mathbb{P}(S)$, with the subset relation, *i.e.*, $(\mathbb{P}(S), \subseteq)$, is a complete lattice.

For ordered sets $(S, \leq)$ and $(T, \leq)$, a function $f : S \to T$ is *monotone* if and only if for all $x, y \in S$ with $x \leq y$, also $f(x) \leq f(y)$.

**Definition 2.7.** Let $(S, \leq)$ be a lattice, and $f : S \to S$ be a function. Element $x \in S$ is a *fixed point* of $f$ if $f(x) = x$.

We have the following key properties of monotone functions over complete lattices. Note that the history of these properties is interesting in its own right, and is nicely described by Lassez *et al.* in [LNS82]. These results were obtained by Knaster and Tarski [Kna28; Tar55].

**Theorem 2.8** (Knaster-Tarski)**.** *Let $(S, \leq)$ be a complete lattice, and $f : S \to S$ a monotone function. Let $F$ be the set of all fixed points of $f$. Then:*

- *$F \neq \emptyset$, i.e., a fixed point exists,*
- *$(F, \leq)$ is a complete lattice,*
- *the least fixed point is $\mu X.f(X) \triangleq \bigwedge \{x \in S \mid f(x) \leq x\}$, and*
- *the greatest fixed point is $\nu X.f(X) \triangleq \bigvee \{x \in S \mid x \leq f(x)\}$.*

Throughout this thesis we write $\sigma$ to denote an arbitrary fixed point sign $\mu$ or $\nu$. In a constructive way, fixed points of monotone functions can be obtained using fixed point iteration starting from the least or greatest element of $S$, *i.e.*, $\bot$ or $\top$. If the domains $S$ are finite, an iteration scheme based on mathematical induction suffices.

**Definition 2.9.** Let $(S, \leq)$ be a complete lattice, with a finite domain $S$, and let $f : S \to S$ be a monotone function. Then we define the $\sigma^n$-approximant, with $n$ a natural number, inductively as follows:

- $\mu^0 X.f(X) \triangleq \bot$,
- $\nu^0 X.f(X) \triangleq \top$, and
- $\sigma^{n+1} X.f(X) \triangleq f(\sigma^n X.f(X))$.

For finite domains, the use of fixed point iteration for computing the fixed point is formalised in the following proposition.

**Proposition 2.10.** *Let $(S, \leq)$ be a complete lattice, with finite domain S, and let $f : S \to S$ be a monotone function. Then*

- $\mu X.f(X) = \bigvee_{n \in \mathbb{N}} \mu^n X.f(X)$, *and*
- $\nu X.f(X) = \bigwedge_{n \in \mathbb{N}} \nu^n X.f(X)$.

In this thesis, especially in Chapter 6, we consider domains that may be uncountably large; hence we need to resort to transfinite induction instead of the ordinary mathematical induction. The following is a generalisation of Definition 2.9.

**Definition 2.11.** Let $(S, \leq)$ be a complete lattice, with a possibly infinite domain $S$, and let $f : S \to S$ be a monotone function. Then we define the $\sigma^\alpha$-approximant, with $\alpha$ an ordinal and $\lambda$ a limit ordinal, by transfinite induction as follows:

- $\mu^0 X.f(X) \triangleq \bot$,
- $\nu^0 X.f(X) \triangleq \top$,
- $\sigma^{\alpha+1} X.f(X) \triangleq f(\sigma^\alpha X.f(X))$,
- $\mu^\lambda X.f(X) \triangleq \bigvee_{\alpha<\lambda} \mu^\alpha X.f(X)$, and
- $\nu^\lambda X.f(X) \triangleq \bigwedge_{\alpha<\lambda} \nu^\alpha X.f(X)$.

We can again use this approximation scheme to compute the fixed points.

**Proposition 2.12.** *For a complete lattice $(S, \leq)$ and a monotone function $f : S \to S$, with Ord the class of all ordinals, the fixed points can also be characterised as follows:*

- $\mu X.f(X) = \bigvee_{\alpha \in Ord} \mu^\alpha X.f(X)$, *and*
- $\nu X.f(X) = \bigwedge_{\alpha \in Ord} \nu^\alpha X.f(X)$.

*Also, there exists an ordinal $\alpha$ with cardinality at most the cardinality of S such that for all ordinals $\beta \geq \alpha$:*

- $\mu X.f(X) = \mu^\beta X.f(X)$, *and dually*

- $\nu X.f(X) = \nu^{\beta} X.f(X)$.

Lattices on sets can be extended to lattices on functions. An order on functions $S \to T$ is inherited from the codomain $T$, *i.e.*, for $f, g : S \to T$, $f \leq g$ if and only if for all $x \in S$, $f(x) \leq g(x)$.

If $S$ and $T$ are complete lattices, then also the set of functions $S \to T$ is a complete lattice, and the supremum and infimum are obtained pointwise.

## 2.4 Modal $\mu$-calculus

Desired properties that a behavioural specification should adhere to are typically expressed using formulae in a temporal logic. For linear-time properties LTL [Pnu77] is a popular choice. For branching-time properties a logic like CTL [CE82] can be used. The logic CTL$^*$ [EH86] subsumes both logics.

The *propositional modal $\mu$-calculus*, originally defined by Kozen [Koz83], again subsumes CTL$^*$. It is a highly-expressive language for analysing behaviours that are defined through a labelled transition system, that extends propositional modal logic with least- and greatest fixed point operators. We first present its syntax.

**Definition 2.13.** We assume the existence of a sufficiently large, countable set of proposition variables $\mathcal{X}^{\sim}$. Let *Act* be a finite set of actions. The set of modal $\mu$-calculus formulae is defined through the following grammar, which is given directly in positive form:

$$\phi, \psi ::= \mathsf{true} \mid \mathsf{false} \mid \tilde{X} \mid \phi \wedge \psi \mid \phi \vee \psi \mid [A]\phi \mid \langle A \rangle \phi \mid \nu \tilde{X}.\phi \mid \mu \tilde{X}.\phi$$

where $\tilde{X} \in \mathcal{X}^{\sim}$ is a proposition variable; $A \subseteq Act$ is a set of actions; $\mu$ is a least fixed point sign and $\nu$ is a greatest fixed point sign.

Here the modal operators bind stronger than the binary operators, *i.e.* $[A]\phi \wedge \psi$ is interpreted as $([A]\phi) \wedge \psi$, and likewise for $\langle A \rangle \phi$ and $\vee$. The binary operators bind stronger than the fixed points, *i.e.*, $\mu \tilde{X}.\phi \wedge \psi$ is interpreted as $\mu \tilde{X}.(\phi \wedge \psi)$, and likewise for $\nu$ and $\vee$.

In standard expositions of the $\mu$-calculus operators $[a]\phi$ and $\langle a \rangle \phi$ are used instead of the generalised modal operators $[A]\phi$ and $\langle A \rangle \phi$ that we present here. Our use of the generalised modal operators is merely for reasons of notational convenience, and has no implications for the presented theory in this thesis. They could easily be defined in terms of the standard modal operators as follows:

$$[A]\phi \triangleq \bigwedge_{a \in A} [a]\phi \qquad\qquad \langle A \rangle \phi \triangleq \bigvee_{a \in A} \langle a \rangle \phi$$

where $\bigwedge_{a \in A} [a]\phi$ is $\mathsf{true}$ if $A = \emptyset$, and likewise, $\bigvee_{a \in A} \langle a \rangle \phi$ is $\mathsf{false}$ if $A = \emptyset$. Henceforth, we write $[a]\phi$ instead of $[\{a\}]\phi$ and $[\overline{a}]\phi$ instead of $[Act \setminus \{a\}]\phi$.

We give a few examples of $\mu$-calculus formulae and their intuitive meaning in the following example. Validity of $\mu$-calculus formulae is illustrated at the end of this section.

**Example 2.14.** Let $L = \langle S, Act, \rightarrow \rangle$ be an LTS. Absence of deadlock is expressed using

$$\nu X.[Act]X \wedge \langle Act \rangle \mathsf{true}$$

The following expression illustrates that the formula $f$ holds in every state of the system:

$$\nu X.[Act]X \wedge f$$

Absence of deadlock is simply an instance of this, where $f = \langle Act \rangle \mathsf{true}$ expresses that at least one transition is enabled. The formula

$$[r](\nu X.\mu Y.([s]X \wedge [\overline{s}]Y))$$

expresses that after every $r$ action all non-$s$ sub-paths are finite. Typical fairness properties are more complicated. The following property states that invariantly, if an $r$ action is enabled infinitely often, then it is also taken infinitely often.

$$\nu W.[Act]W \wedge \nu X.\mu Y.\nu Z([r]X \wedge ([r]\mathsf{false} \vee [\overline{r}]Y) \wedge [\overline{r}]Z)$$

The set of *occurring propositional variables*, *i.e.* the proposition variables that syntactically occur in $\phi$, is denoted $\mathsf{occ}(\phi)$ , and defined as:

$$\begin{aligned}
\mathsf{occ}(\mathsf{true}) &\triangleq \emptyset & \mathsf{occ}(\phi \wedge \psi) &\triangleq \mathsf{occ}(\phi) \cup \mathsf{occ}(\psi) \\
\mathsf{occ}(\mathsf{false}) &\triangleq \emptyset & \mathsf{occ}(\phi \vee \psi) &\triangleq \mathsf{occ}(\phi) \cup \mathsf{occ}(\psi) \\
\mathsf{occ}(\tilde{X}) &\triangleq \{\tilde{X}\} & \mathsf{occ}(\mu\tilde{X}.\phi) &\triangleq \mathsf{occ}(\phi) \setminus \{\tilde{X}\} \\
\mathsf{occ}([A]\phi) &\triangleq \mathsf{occ}(\phi) & \mathsf{occ}(\nu\tilde{X}.\phi) &\triangleq \mathsf{occ}(\phi) \setminus \{\tilde{X}\} \\
\mathsf{occ}(\langle A \rangle \phi) &\triangleq \mathsf{occ}(\phi)
\end{aligned}$$

The set of *bound proposition variables* is defined inductively as follows.

$$\begin{aligned}
\mathsf{bnd}(\mathsf{true}) &\triangleq \emptyset & \mathsf{bnd}(\phi \wedge \psi) &\triangleq \mathsf{bnd}(\phi) \cup \mathsf{bnd}(\psi) \\
\mathsf{bnd}(\mathsf{false}) &\triangleq \emptyset & \mathsf{bnd}(\phi \vee \psi) &\triangleq \mathsf{bnd}(\phi) \cup \mathsf{bnd}(\psi) \\
\mathsf{bnd}(\tilde{X}) &\triangleq \emptyset & \mathsf{bnd}(\mu\tilde{X}.\phi) &\triangleq \mathsf{bnd}(\phi) \cup \{\tilde{X}\} \\
\mathsf{bnd}([A]\phi) &\triangleq \mathsf{bnd}(\phi) & \mathsf{bnd}(\nu\tilde{X}.\phi) &\triangleq \mathsf{bnd}(\phi) \cup \{\tilde{X}\} \\
\mathsf{bnd}(\langle A \rangle \phi) &\triangleq \mathsf{bnd}(\phi)
\end{aligned}$$

With $\mathsf{free}(\phi) \triangleq \mathsf{occ}(\phi) \setminus \mathsf{bnd}(\phi)$ we refer to the set of *free proposition variables*.

Formula $\phi$ is said to be *closed* if and only if $\mathsf{free}(\phi) = \emptyset$. We only consider $\mu$-calculus formulae $\phi$ that are *well-formed*, *i.e.*:

1. there are no two distinct subformulae of $\phi$ that bind the same proposition variable, formally for all $\sigma_1\tilde{X}_1.\psi_1$ and $\sigma_2\tilde{X}_2.\psi_2$ that are distinct subformulae of $\phi$, $\tilde{X}_1 \neq \tilde{X}_2$;
2. for every free proposition variable $\tilde{X} \in \mathsf{free}(\phi)$, no subformula $\sigma\tilde{X}.\psi$ (binding $\tilde{X}$ locally) occurs in $\phi$.

The well-formedness requirement is a technicality and does not incur a loss of generality of the theory.

Modal $\mu$-calculus formulae $\phi$ are *interpreted* in the context of a labelled transition system $\langle S, Act, \rightarrow \rangle$ and an *environment* $\theta : \mathcal{X}^{\tilde{}} \rightarrow \mathbb{P}(S)$ that assigns sets of states to proposition variables. We write $\theta[\tilde{X} := S']$ to represent the environment in which $\tilde{X}$ receives the value $S'$, and all other proposition variables have values that coincide with those given by $\theta$.

**Definition 2.15.** Let $L = \langle S, Act, \rightarrow \rangle$ be a labelled transition system and let $\theta : \mathcal{X}^{\tilde{}} \rightarrow \mathbb{P}(S)$ be a proposition environment. The semantics of a $\mu$-calculus formula $\phi$ is defined inductively as follows:

$$
\begin{array}{lcl}
[\![\mathsf{true}]\!]\theta & \triangleq & S \\
[\![\mathsf{false}]\!]\theta & \triangleq & \emptyset \\
[\![\tilde{X}]\!]\theta & \triangleq & \theta(\tilde{X}) \\
[\![\phi \wedge \psi]\!]\theta & \triangleq & [\![\phi]\!]\theta \cap [\![\psi]\!]\theta \\
[\![\phi \vee \psi]\!]\theta & \triangleq & [\![\phi]\!]\theta \cup [\![\psi]\!]\theta \\
[\![[A]\phi]\!]\theta & \triangleq & \{s \in S \mid \forall s' \in S : \forall a \in A : \; s \xrightarrow{a} s' \implies s' \in [\![\phi]\!]\theta\} \\
[\![\langle A \rangle \phi]\!]\theta & \triangleq & \{s \in S \mid \exists s' \in S : \exists a \in A : \; s \xrightarrow{a} s' \wedge s' \in [\![\phi]\!]\theta\} \\
[\![\nu \tilde{X}.\phi]\!]\theta & \triangleq & \nu T \in \mathbb{P}(S).[\![\phi]\!]\theta[\tilde{X} := T] \\
[\![\mu \tilde{X}.\phi]\!]\theta & \triangleq & \mu T \in \mathbb{P}(S).[\![\phi]\!]\theta[\tilde{X} := T]
\end{array}
$$

Note that for the fixed point formulae the semantics is defined through a predicate transformer from sets of states to sets of states, of which we consider the fixed point.

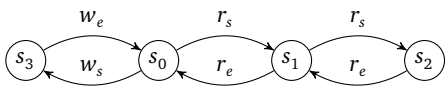
The *global* model checking problem, denoted $L, \theta \models \phi$, is to answer for all states $s \in S$ of a given labelled transition system $L = \langle S, Act, \rightarrow \rangle$, whether or not $s \in [\![\phi]\!]\theta$, for given formula $\phi$ and environment $\theta$. The *local* model checking problem, denoted $L, s, \theta \models \phi$, is the problem whether $s \in [\![\phi]\!]\theta$ for a given state $s \in S$. Often, one is only interested in *closed* formulae, in which $\theta$ is immaterial to the solution; in these cases we typically omit $\theta$, and write $L, s \models \phi$ instead. Small examples of model checking problems can be found throughout the remainder of this thesis.

There is a close correspondence between bisimulation and the $\mu$-calculus, which is formalised in the following theorem.

**Theorem 2.16.** *Let $L = \langle S, Act, \rightarrow \rangle$, be a finitely branching labelled transition system with $s, s' \in S$. Then $s \leftrightarrows s'$ if and only if for all closed modal $\mu$-calculus formulae $\phi$, $L, s \models \phi \iff L, s' \models \phi$.*

An obvious strategy for solving a typical model checking problem is through the use of the fixed point approximation schemes for computing the solution to the fixed points of monotone operators in a complete lattice as given in Definition 2.9. We demonstrate this approach using the following example.

**Example 2.17.** Recall the labelled transition system from Example 2.2 (depicted below), modelling mutual exclusion between two readers and a single writer.




The verification problem we consider is $\nu\tilde{X}.\mu\tilde{Y}.\ \langle r_s\rangle\tilde{X} \vee \langle\overline{r_s}\rangle\tilde{Y}$, modelling that on some path, a reader can infinitely often start reading. We illustrate the solution of this model checking problem using fixed point iteration. We give the approximants.

$$
\begin{aligned}
X_0 &= \{s_0, s_1, s_2, s_3\} \\
X_1 &= Y_1 && Y_{1,0} = \emptyset \\
& && Y_{1,1} = \{s_0, s_1\} \\
& && Y_{1,2} = \{s_0, s_1, s_2, s_3\} \\
&= \{s_0, s_1, s_2, s_3\}
\end{aligned}
$$

Initially, since $X$ is a greatest fixed point, we assume the property holds for all states ($X_0$). For $X_1$ we use $X_0$ to compute the nested occurrence of $Y$, $Y_{1,0}$ is, initially, the empty set due to the least fixed point. We now iterate until we reach a fixed point. In $Y_{1,1}$, observe that $s_0$ and $s_1$ can do an $r_s$ step to states in $X_0$. For $Y_{1,2}$, the $s_3$ can do a $w_e$ step to state $s_0$ and $s_2$ can do an $r_e$ step to $s_1$. Since both $s_0$ and $s_1$ are in $Y_{1,1}$ we also add $s_2$ and $s_3$ in this step. Since we now added all vertices, the computation for $Y$ is stable. Observe that $X_1 = X_0$, so also the computation for $X$ is stable, and we find that the property holds in all states of the system.

In this example, using fixed point iteration for solving the $\mu$-calculus model checking problem works efficiently. In general, however, this approach is naive, since innermost fixed points are recomputed in every iteration of the outermost fixed point, even if the innermost fixed point does not depend on the current approximation of the outermost fixed point. More efficient solutions that avoid unnecessary recomputations have been described in the literature by, *e.g.*, Emerson and Lei [EL86].

## 2.5 Boolean Equation Systems

A *Boolean equation system* [Lar93; Mad97] is a sequence of fixed point equations, in which all equations range over the Boolean lattice. The interest in equation systems has both practical and theoretical origins. On the practical side, equation systems have been used as a uniform framework for solving traditional verification problems. The *model checking* problem of several variations and sublogics of the $\mu$-calculus have been mapped on the Boolean equation systems. This varies from the alternation free fragment of the $\mu$-calculus [Dic86], to the full modal $\mu$-calculus [Mad97], and several subclasses of intermediate complexity.

Furthermore checking of a large number of behavioural equivalences and preorders has been mapped to BESs. This ranges from the prebisimulation- [CS91] and simulation preorder [Lar93], to strong bisimulation [Lar93; Mat03], branching bisimulation [VL94; Mat03], observational, $\tau^*.a$ and safety equivalence [Mat03]. For more contemporary descriptions of these problems the reader is also referred to [Mat06; Che+07]. All of

these problems can be mapped onto Boolean equation systems with at most two blocks of equations with different fixed point signs.

Computation of fixed points over systems of equations has a rich history in the context of verification. Dicky expresses properties as least fixed points over systems of mutually recursive equations [Dic86]. The algorithm that is presented in *ibid.* runs in time quadratic in both the size of the transition system and the size of the equation system. Arnold and Crubille improved upon this algorithm in [AC88], by presenting an algorithm that is linear in the size of the transition system, yet quadratic in the size of the equation system. Vergauwen and Lewi [VL92] finally gave an algorithm for solving alternation free equation systems in time linear in both the size of the transition system and the equation system. A similar improvement was presented independently by Cleaveland and Steffen in [CS91]. In [VL94] Vergauwen and Lewi present an algorithm that works for Boolean equation systems with two blocks of like-signed equations, but of which one of the described restore strategies was shown to be incorrect by Liu and Smolka [LS98, Appendix A].

Larsen presented a local algorithm for alternation free BESs that runs in time quadratic in the size of the BES [Lar93]. *Boolean Graphs* are introduced in [And94], in an attempt to use graphs for representing the (implicit) equation systems (in simple form), underlying model checking problems obtained by verifying $\mu$-calculus formulae on state spaces. Equations are represented by vertices, and dependencies on variables are represented by the edges. In addition, each vertex is labelled with either $\vee$ or $\wedge$, representing the fact that the right-hand side of the equation is disjunctive or conjunctive, respectively. On the basis of the graph representation, Andersen describes a global model checking algorithm for alternation-free equation systems that runs in linear time. In the same paper, he also generalises the algorithm to equation systems underlying the full modal $\mu$-calculus. Also, Andersen shows a local, on-the-fly algorithm for alternation-free equation systems that improves over the running time complexity Larsen's approach.

The on-the-fly techniques by Andersen are generalised to the full modal $\mu$-calculus in [LRS98]. The graphs underlying the latter approach, called *Partitioned Dependency Graphs*, generalise Andersen's Boolean Graphs, by considering *hyper-edges* from vertices to sets of vertices. Liu and Smolka in [LS98] propose an improvement over this approach for the special case of alternation-free equation systems, using *dependency graphs*. The latter simplify the Partitioned Dependency Graphs, and, at the same time, generalise the Boolean Graphs of Andersen, giving rise to simpler equation system resolution algorithms. In addition, Liu and Smolka show that their dependency graphs are useful for solving Horn clauses.

During the past decade research has again focussed on the solving of subclasses of Boolean equation systems. Mateescu presented several results on alternation-free BESs [Mat03; Mat06; MO08b; MO08a]. Keinänen extends the Boolean Graphs of Andersen by decorating each vertex, in addition to the labelling with $\wedge$ or $\vee$, with a natural number that abstractly represents the fixed point sign of the equation, see [Kei06]. These graphs are also referred to as *dependency graphs*. Groote and Keinänen use the dependency graphs to give efficient algorithms for solving conjunctive and disjunctive equation systems with alternating fixed points [GK05]. Recently Kumar, Ramakrishnan and Smolka showed that the solutions of BESs can be characterised in terms of models of logic programs [KRS01].

### 2.5.1 Theory of Boolean Equation Systems

As far as we are aware, the presentation of Boolean equation systems that we use in this thesis is due to Mader [Mad97], who also presented an algorithm for globally solving Boolean equation systems that is akin to Gauß elimination, see Lemmata 2.25 and 2.26.

**Definition 2.18.** Assume the existence of a sufficiently large, countable set of proposition variables $\mathscr{X}$, disjoint from $\tilde{\mathscr{X}}$. A *Boolean equation system (BES)* $\mathscr{E}$ is defined by the following grammar:

$$\begin{array}{lcl} \mathscr{E} & ::= & \epsilon \mid (\nu X = f)\ \mathscr{E} \mid (\mu X = f)\ \mathscr{E} \\ f, g & ::= & \mathsf{true} \mid \mathsf{false} \mid X \mid f \wedge g \mid f \vee g \end{array}$$

where $\epsilon$ is the empty BES; $X \in \mathscr{X}$ is a proposition variable; and $f, g$ are proposition formulae. We sometimes write $b, c$ for arbitrary Boolean constants, *i.e.*, $b, c \in \{\mathsf{true}, \mathsf{false}\}$.

We only consider equation systems that are *well-formed*, *i.e.*, equation systems $\mathscr{E}$, in which a proposition variable $X$ occurs at the left-hand side of at most a single equation in $\mathscr{E}$.

Semantically, a Boolean equation system is a fixed point equation system over the Boolean lattice $(\mathbb{B}, \sqsubseteq)$ where $f \sqsubseteq g$ if $f$ implies $g$. Proposition formulae are interpreted in the context of an *environment* $\eta \colon \mathscr{X} \to \mathbb{B}$, which assigns a Boolean value to every proposition variable. The ordering $\sqsubseteq$ on environments is defined as $\eta \sqsubseteq \eta'$ if and only if $\eta(X) \sqsubseteq \eta'(X)$ for all $X$. For reading ease, we do not formally distinguish between a semantical Boolean value and its representation by $\mathsf{true}$ and $\mathsf{false}$; likewise, for the operands $\wedge$ and $\vee$.

**Definition 2.19** ([Mad97])**.** Let $\eta \colon \mathscr{X} \to \mathbb{B}$ be an environment. The *interpretation* $[\![ f ]\!]\eta$ maps a proposition formula $f$ to $\mathsf{true}$ or $\mathsf{false}$:

$$\begin{array}{lll} [\![ X ]\!]\eta \triangleq \eta(X) & & \\ [\![ \mathsf{true} ]\!]\eta \triangleq \mathsf{true} & & [\![ f \wedge g ]\!]\eta \triangleq [\![ f ]\!]\eta \wedge [\![ g ]\!]\eta \\ [\![ \mathsf{false} ]\!]\eta \triangleq \mathsf{false} & & [\![ f \vee g ]\!]\eta \triangleq [\![ f ]\!]\eta \vee [\![ g ]\!]\eta \end{array}$$

We define *logical equivalence* between to proposition formulae $f, g$, denoted $f \equiv g$, as $[\![ f ]\!]\eta = [\![ g ]\!]\eta$ for all $\eta$.

The *solution of a BES*, given an environment $\eta$, is inductively defined as follows:

$$\begin{array}{lcl} [\![ \epsilon ]\!]\eta & \triangleq & \eta \\ [\![ (\sigma X = f)\ \mathscr{E} ]\!]\eta & \triangleq & \begin{cases} [\![ \mathscr{E} ]\!](\eta[X := [\![ f ]\!]([\![ \mathscr{E} ]\!]\eta[X := \mathsf{false}])]) & \text{if } \sigma = \mu \\ [\![ \mathscr{E} ]\!](\eta[X := [\![ f ]\!]([\![ \mathscr{E} ]\!]\eta[X := \mathsf{true}])]) & \text{if } \sigma = \nu \end{cases} \end{array}$$

A solution to an equation system verifies every equation, in the sense that the value at the left-hand side is logically equivalent to the value at the right-hand side of the equation. At the same time, the fixed point signs of left-most equations *outweigh* the fixed point signs of those equations that follow, *i.e.*, the fixed point signs of left-most equations are more important. The latter phenomenon is a result of the nested recursion for evaluating the proposition $f$ of the left-most equation $(\sigma X = f)$, assuming an extremal value for $X$. As a consequence, the solution is order-sensitive as indicated in the following example.

**Example 2.20.** Consider equation systems $\mathscr{E}_1 = (\nu X = Y)(\mu Y = X)$, and $\mathscr{E}_2 = (\mu X = Y)(\nu Y = X)$. Note that the fixed point symbols in the equations have been exchanged. In $\mathscr{E}_1$ the solution for both $X$ and $Y$ is true, because the greatest fixed point sign got priority, and prefers true as a solution, whereas in $\mathscr{E}_2$ the solution for both variables is false due to the least fixed point in the equation for $X$.

It is exactly the tree-like recursion in Definition 2.19 that makes the solution concept of BESs intricately complex.

If no disjunctions occur in the right-hand sides of the equations in a BES, we say that the BES is in *conjunctive form*, likewise, if no conjunctions occur the BES is in *disjunctive form*. For every equation system, we can construct equation systems in conjunctive and disjunctive form with the same solution. This approach is based on a purely syntactic ordering on formulae and equation systems. We first introduce this ordering.

**Definition 2.21** ([Mad97, Definition 3.15])**.** Let $n \in \mathbb{N}$, $\mathscr{E} = (\sigma_1 X_1 = f_1) \ldots (\sigma_n X_n = f_n)$, and $\mathscr{F} = (\sigma_1 X_1 = g_1) \ldots (\sigma_n X_n = g_n)$, then $\mathscr{E} \Subset \mathscr{F} \stackrel{\Delta}{=} f_i \sqsubseteq g_i$ for all $1 \leq i \leq n$.

The syntactic ordering on BESs, and the ordering on environments are related as follows.

**Lemma 2.22** ([Mad97, Lemma 3.16])**.** *Let $\mathscr{E}$ and $\mathscr{F}$ be equation systems, then $\mathscr{E} \Subset \mathscr{F}$ implies for all environments $\eta$, $[\![\mathscr{E}]\!]\eta \sqsubseteq [\![\mathscr{F}]\!]\eta$.*

The ordering on equation systems gives rise to the following proposition, that non-constructively shows that for every BES there exist equation systems in conjunctive and disjunctive form with the same solution.

**Proposition 2.23** ([Mad97, Proposition 3.36])**.** *Let $\mathscr{E}$ be a Boolean equation system, and let $\eta$ be an environment. There exist Boolean equation systems $\mathscr{E}', \mathscr{E}''$ such that:*

- $[\![\mathscr{E}]\!]\eta = [\![\mathscr{E}']\!]\eta = [\![\mathscr{E}'']\!]\eta$,
- $\mathscr{E}' \Subset \mathscr{E} \Subset \mathscr{E}''$, *and*
- $\mathscr{E}'$ *and* $\mathscr{E}''$ *are in conjunctive form and disjunctive form, respectively.*

Since we can do this for every BES, we can also find equation systems in which every right hand side is neither conjunctive nor disjunctive.

**Corollary 2.24** ([Mad97, Corollary 3.37])**.** *Let $\mathscr{E}$ be a Boolean equation system, and let $\eta$ be an environment. There exists a Boolean equation system $\mathscr{F}$ such that $[\![\mathscr{E}]\!]\eta = [\![\mathscr{F}]\!]\eta$ and $\mathscr{F}$ is derived from $\mathscr{E}$ by selecting one variable of the right hand side in every equation.*

For equation systems in which every right hand side contains occurrences of at most one binary connective, the above boils down to selecting one conjunct for every equation or one disjunct for every equation, depending on whether we construct a conjunctive or a disjunctive equation system in Proposition 2.23.

Equation systems can be solved using Gauß elimination, which combines *elimination steps* and *substitution steps*. This method was defined by Mader [Mad97] through the following two lemmata.

In an elimination step, the right hand side of a single equation is simplified based on the following lemma.

**Lemma 2.25** ([Mad97, Lemma 6.2]). *Let $\mathscr{E}_1, \mathscr{E}_2$ be Boolean equations, $X$ a proposition variable, and $f$ be a propositional formula, then for all environments $\eta$*

$$\begin{aligned}[\![\mathscr{E}_1(\nu X = f)\mathscr{E}_2]\!]\eta &= [\![\mathscr{E}_1(\nu X = f[X := \mathsf{true}])\mathscr{E}_2]\!]\eta \\ [\![\mathscr{E}_1(\mu X = f)\mathscr{E}_2]\!]\eta &= [\![\mathscr{E}_1(\mu X = f[X := \mathsf{false}])\mathscr{E}_2]\!]\eta\end{aligned}$$

In the substitution step, the right hand side of an equation in the BES is substituted for its left hand side in an equation occurring *before* it in the equation system. This is based on the following lemma.

**Lemma 2.26** ([Mad97, Lemma 6.3]). *Let $\mathscr{E}_1$, $\mathscr{E}_2$ and $\mathscr{E}_3$ be Boolean equation systems, then for all environments $\eta$*

$$[\![\mathscr{E}_1(\sigma_X X = f)\mathscr{E}_2(\sigma_Y Y = g)\mathscr{E}_3]\!]\eta = [\![\mathscr{E}_1(\sigma_X X = f[Y := g])\mathscr{E}_2(\sigma_Y Y = g)\mathscr{E}_3]\!]\eta.$$

In line with the notions of bound and occurring proposition variables for $\mu$-calculus formulae, we introduce analogous notions for equation systems. Let $\mathscr{E}$ be an arbitrary equation system. The set of *bound proposition variables* of $\mathscr{E}$, denoted $\mathsf{bnd}(\mathscr{E})$, is the set of variables occurring at the left-hand side of the equations in $\mathscr{E}$. The set of *occurring proposition variables*, denoted $\mathsf{occ}(\mathscr{E})$, is the set of variables occurring at the right-hand side of some equation in $\mathscr{E}$. We define them formally as follows.

$$\begin{aligned}\mathsf{bnd}(\epsilon) &\triangleq \emptyset & \mathsf{bnd}((\sigma X = f)\ \mathscr{E}) &\triangleq \mathsf{bnd}(\mathscr{E}) \cup \{X\} \\ \mathsf{occ}(\epsilon) &\triangleq \emptyset & \mathsf{occ}((\sigma X = f)\ \mathscr{E}) &\triangleq \mathsf{occ}(\mathscr{E}) \cup \mathsf{occ}(f)\end{aligned}$$

where $\mathsf{occ}(f)$ is defined inductively as follows:

$$\begin{aligned}\mathsf{occ}(c) &\triangleq \emptyset & \mathsf{occ}(X) &\triangleq \{X\} \\ \mathsf{occ}(f \vee g) &\triangleq \mathsf{occ}(f) \cup \mathsf{occ}(g) & \mathsf{occ}(f \wedge g) &\triangleq \mathsf{occ}(f) \cup \mathsf{occ}(g)\end{aligned}$$

For an equation $\sigma X = f$, we write $\varphi_X$ to denote its right hand side, in this case $\varphi_X = f$. We write $\mathsf{occ}(\varphi_X)$ to denote the set of variables occurring in the right hand side of the defining equation of $X$. The free proposition variables of an equation system $\mathscr{E}$ are defined as $\mathsf{free}(\mathscr{E}) \triangleq \mathsf{occ}(\mathscr{E}) \setminus \mathsf{bnd}(\mathscr{E})$.

We say an equation system $\mathscr{E}$ is *closed* whenever $\mathsf{occ}(\mathscr{E}) \subseteq \mathsf{bnd}(\mathscr{E})$. Intuitively, a (closed) equation system uniquely assigns truth values to its bound proposition variables. Closed equation systems enjoy the property that the solution to the equation system is independent of the environment in which it is defined, *i.e.*, for all environments $\eta, \eta'$, we have $[\![\mathscr{E}]\!]\eta(X) = [\![\mathscr{E}]\!]\eta'(X)$ for all $X \in \mathsf{bnd}(\mathscr{E})$. For this reason, we henceforth often refrain from writing the environment explicitly in our considerations dealing with closed equation systems, *i.e.*, we write $[\![\mathscr{E}]\!]$, and $[\![\mathscr{E}]\!](X)$ instead of the more verbose $[\![\mathscr{E}]\!]\eta$ and $[\![\mathscr{E}]\!]\eta(X)$.

If the variables in two equation systems are disjoint we call them compatible.

**Definition 2.27.** Let $\mathscr{E}$ and $\mathscr{F}$ be Boolean equation systems. Then $\mathscr{E}$ and $\mathscr{F}$ are compatible if and only if

$$\mathsf{bnd}(\mathscr{E}) \cap \mathsf{bnd}(\mathscr{F}) = \mathsf{bnd}(\mathscr{E}) \cap \mathsf{occ}(\mathscr{F}) = \mathsf{occ}(\mathscr{E}) \cap \mathsf{bnd}(\mathscr{F}) = \emptyset.$$

Compatible equation systems may be combined. This is formalised by the following lemma.

**Lemma 2.28** ([Mad97, Lemma 3.10])**.** *Let $\mathscr{E}$ and $\mathscr{F}$ be compatible BESs, then for all environments $\eta$, $[\![\mathscr{E}]\!][\![\mathscr{F}]\!]\eta = [\![\mathscr{E}\mathscr{F}]\!]\eta$.*

The following lemma relates the semantics for open equation systems to that of closed equation systems. We write $\mathscr{E}[X := b]$, where $X \notin \mathsf{bnd}(\mathscr{E})$ and $b \in \{\mathsf{true}, \mathsf{false}\}$ is a constant, to denote the equation system in which each syntactic occurrence of $X$ is replaced by $b$.

**Lemma 2.29.** *Let $\mathscr{E}$ be an equation system, and let $\eta$ be an arbitrary environment. Assume $X \notin \mathsf{bnd}(\mathscr{E})$ is a proposition variable, and let $b$ be such that $\eta(X) = [\![b]\!]$. Then $[\![\mathscr{E}]\!]\eta = [\![\mathscr{E}[X := b]]\!]\eta$.*

*Proof.* We show this by induction on the size of $\mathscr{E}$. The base case for $\mathscr{E} = \epsilon$ follows immediately. As our induction hypothesis, we take

$$\forall \eta, b, X \notin \mathsf{bnd}(\mathscr{E}) : \ [\![b]\!] = \eta(X) \implies [\![\mathscr{E}]\!]\eta = [\![\mathscr{E}[X := b]]\!]\eta \qquad \text{(IH)}$$

Assume our induction hypothesis holds for $\mathscr{E}$, and let $\eta$ and $b$ be such that $[\![b]\!] = \eta(X)$. Consider the equation system $(\nu Y = f)\ \mathscr{E}$, and assume $X \notin \mathsf{bnd}((\nu Y = f)\ \mathscr{E})$. Using the semantics of equation systems, we reason as follows:

$$\begin{array}{ll} & [\![(\nu Y = f)\mathscr{E}]\!]\eta \\ = & [\![\mathscr{E}]\!]\eta[Y := [\![f]\!]([\![\mathscr{E}]\!]\eta[Y := \mathsf{true}])] \\ =^{2\times IH} & [\![\mathscr{E}[X := b]]\!]\eta[Y := [\![f]\!]([\![\mathscr{E}[X := b]]\!]\eta[Y := \mathsf{true}])] \\ =^{\ddagger} & [\![\mathscr{E}[X := b]]\!]\eta[Y := [\![f[X := b]]\!]([\![\mathscr{E}[X := b]]\!]\eta[Y := \mathsf{true}])] \\ = & [\![((\nu Y = f)\ \mathscr{E})\ [X := b]]\!]\eta \end{array}$$

where at $\ddagger$, we have used that $[\![f]\!]\eta = [\![f[X := b]]\!]\eta$ for $[\![b]\!] = \eta(X)$. The case for $(\mu Y = f)\ \mathscr{E}$ follows the exact same line of reasoning and is therefore omitted. □

We define the *free variable closure* of an equation system $\mathscr{E}$, given an environment $\eta$, as the equation system $\mathscr{E}_c$, which is the equation system in which all free predicate variables have been replaced with the syntactic counterpart of the value they are assigned in $\eta$, according to the lemma above. The free variable closure of a formula is defined analogously.

**Corollary 2.30.** *Let $\mathscr{E}$ be a BES, and $\eta$ an environment. Let $\mathscr{E}_c$ be the free variable closure of $\mathscr{E}$ using $\eta$. Then $[\![\mathscr{E}]\!]\eta = [\![\mathscr{E}_c]\!]$.*

The *rank* of a proposition variable $X \in \mathsf{bnd}(\mathscr{E})$, notation $\mathsf{rank}_{\mathscr{E}}(X)$, is defined as follows:

$$\mathsf{rank}_{(\sigma Y = f)\mathscr{E}}(X) \triangleq \begin{cases} \mathsf{rank}_{\mathscr{E}}(X) & \text{if } X \neq Y \\ \mathscr{E}\rceil\sigma & \text{otherwise} \end{cases}$$

where $\mathscr{E}\rceil\sigma$ is defined as:

$$\epsilon\rceil\sigma \triangleq \begin{cases} 0 & \text{if } \sigma = \nu \\ 1 & \text{otherwise} \end{cases} \qquad (\sigma' Y = f)\mathscr{E}\rceil\sigma \triangleq \begin{cases} \mathscr{E}\rceil\sigma & \text{if } \sigma = \sigma' \\ 1 + \mathscr{E}\rceil\sigma' & \text{if } \sigma \neq \sigma' \end{cases}$$

Informally, the rank of a variable $X$ is the $i^{th}$ block of like-signed equations, containing $X$'s defining equation, counting from right-to-left and starting at 0 if the last equation is a greatest fixed point sign, and 1 otherwise.

Willemse presented an elegant method to show that the solutions of two equation systems in the more general setting of *parameterised Boolean equation systems* are the same [Wil10]. We introduce this approach in its full generality in Chapter 6. At this point we present a simplified version of this theory, tailored to Boolean equation systems, that was presented by Gazda and Willemse [GW12].

**Definition 2.31** ([GW12])**.** Given a relation $R \subseteq \mathscr{X} \times \mathscr{X}$, an environment $\eta$ is an $R$-correlation if and only if $X\ R\ Y$ implies $\eta(X) = \eta(Y)$.

The set of all $R$-correlating environments is denoted by $\Theta_R$

**Definition 2.32** ([GW12, Definition 4])**.** Let $\mathscr{E}$ be a BES. Relation $R \subseteq \mathscr{X} \times \mathscr{X}$ is a *consistent correlation* if, for $X, Y \in \mathsf{bnd}(\mathscr{E})$, $X\ R\ Y$ implies:

- $\mathsf{rank}_{\mathscr{E}}(X) = \mathsf{rank}_{\mathscr{E}}(Y)$, and
- for all $\eta \in \Theta_R$, $[\![\varphi_X]\!]\eta = [\![\varphi_Y]\!]\eta$

We say that $X, Y \in \mathsf{bnd}(\mathscr{E})$ consistently correlate if and only if $X\ R\ Y$ for some consistent correlation $R \subseteq \mathsf{bnd}(\mathscr{E}) \times \mathsf{bnd}(\mathscr{E})$.

Consistent correlations allow to prove that the solutions of variables in an equation system coincide.

**Theorem 2.33** ([GW12, Theorem 1])**.** *Let $\mathscr{E}$ be a BES, and let $R \subseteq \mathscr{X} \times \mathscr{X}$ be a consistent correlation on $\mathscr{E}$, then for all $R$-correlating environments $\eta$, $[\![\mathscr{E}]\!]\eta$ is also a correlating environment.*

For some purposes it is convenient to work with equation systems in a restricted syntax. Among others, we use this syntax to relate parity games and Boolean equation systems in the next section.

**Definition 2.34.** Let $\mathscr{E}$ be an equation system. $\mathscr{E}$ is in *simple recursive form (SRF)* if the right-hand sides $f$ of every one of its equations can be written using the following grammar:

$$f ::= X \mid \bigvee F \mid \bigwedge F, \quad \text{where } F \subseteq \mathscr{X}, \text{ with } |F| > 0.$$

where the interpretation is given by the following rules:

$$[\![X]\!]\eta \triangleq \eta(X) \qquad [\![\textstyle\bigvee F]\!]\eta \triangleq \bigvee\{\eta(X) \mid X \in F\} \qquad [\![\textstyle\bigwedge F]\!]\eta \triangleq \bigwedge\{\eta(X) \mid X \in F\}$$

If, in addition, also the Boolean constants $\mathsf{true}$ and $\mathsf{false}$ are allowed in the above syntax, we say that an equation system is in *simple form* [Mad97].

Every closed equation system $\mathscr{E}$ can be rewritten to an equation system $\tilde{\mathscr{E}}$ in SRF such that $[\![\mathscr{E}]\!](X) = [\![\tilde{\mathscr{E}}]\!](X)$ for all $X \in \mathsf{bnd}(\mathscr{E})$, *i.e.*, the transformation to SRF preserves the solution of bound variables. The transformation to SRF allows for some freedom with respect to the relative placement of newly introduced equations. In this thesis we obtain a BES in SRF by repeatedly applying the following transformations:

- Alternating conjunctions and disjunctions in a single right hand side of an equation can be removed by appending an additional equation to the end of the BES.
  - $\mathscr{E}_0(\sigma X = f \wedge g)\mathscr{E}_1$, where $g$ is disjunctive, is replaced by $\mathscr{E}_0(\sigma X = f \wedge X')\mathscr{E}_1(\sigma' X' = g)$, likewise if $f$ is disjunctive, and
  - $\mathscr{E}_0(\sigma X = f \vee g)\mathscr{E}_1$, where $g$ is conjunctive, is replaced by $\mathscr{E}_0(\sigma X = f \vee X')\mathscr{E}_1(\sigma' X' = g)$, likewise if $f$ is conjunctive.
- Constants occurring in the right hand side of an equation may also be removed by introducing additional equations:
  - if $\mathsf{true}$ occurs in the right hand side of some equation in $\mathscr{E}$ it may be removed by replacing $\mathscr{E}$ with $\mathscr{E}[\mathsf{true} := X_{\mathsf{true}}](\nu X_{\mathsf{true}} = X_{\mathsf{true}})$, and
  - if $\mathsf{false}$ occurs in the right hand side of some equation in $\mathscr{E}$ it may be removed by replacing $\mathscr{E}$ with $\mathscr{E}[\mathsf{false} := X_{\mathsf{false}}](\mu X_{\mathsf{false}} = X_{\mathsf{false}})$,

  where $\mathscr{E}[\mathsf{true} := X_{\mathsf{true}}]$ denotes the syntactic replacement of all occurrences of $\mathsf{true}$ in $\mathscr{E}$ with $X_{\mathsf{true}}$, likewise for $\mathsf{false}$.

This transformation leads to a linear blow-up of the original equation system, and its correctness follows from Lemmata 2.25 and 2.26. Note that we fix one definition of the transformation to SRF, however, the lemmata leave room for alternative transformations that may be more suitable in some contexts. Especially the placement of the newly introduced equations may impact the effectiveness of some heuristics on the resulting equation system in SRF.

Finally, we introduce some generic shorthand notation. The operators $\bigsqcap$ and $\bigsqcup$ are used as shorthand for nested applications of $\wedge$ and $\vee$. Formally, these are defined as follows. Let $\lessdot$ be a total order on $\mathscr{X} \cup \{\mathsf{true}, \mathsf{false}\}$. Assuming that $\lessdot$ is lifted to a total ordering on formulae, we define for formula $f$ $\lessdot$-smaller than all formulae in a finite, non-empty set $F$ ($f \notin F$):

$$\bigsqcap \emptyset \triangleq \mathsf{true} \qquad \bigsqcap \{f\} \triangleq f \wedge f \qquad \bigsqcap (\{f\} \cup F) \triangleq f \wedge \left(\bigsqcap F\right)$$

$$\bigsqcup \emptyset \triangleq \mathsf{false} \qquad \bigsqcup \{f\} \triangleq f \vee f \qquad \bigsqcup (\{f\} \cup F) \triangleq f \vee \left(\bigsqcup F\right)$$

Note that the duplication introduced by this definition does not have any semantic influence, *i.e.*, $\bigsqcap F \equiv \bigwedge F$ and $\bigsqcup F \equiv \bigvee F$ for all $F \subseteq \mathscr{X}$. However, compared to the standard operators $\bigwedge$ and $\bigvee$ these newly introduced operators preserve some additional information in case they are applied to a singleton set of variables. We use these operators in the translation of the model checking problem to BESs, and they are instrumental in some of the proofs in Chapter 3. At the cost of a more complex definition, duplication of the least element could be avoided in the case that $F$ contains at least two elements.

We also define an equation system obtained from a set of equations. Let $X = f$ be an equation, where $f$ is a proposition formula and $X$ is a proposition variable. Assuming that $X$ is $\lessdot$-smaller than all left-hand side variables in the equations in a finite set of equations $E$, we define:

$$\sigma\{X = f\} \triangleq (\sigma X = f) \qquad \sigma(\{X = f\} \cup E) \triangleq (\sigma X = f)\sigma E$$

This notation is, *e.g.*, used to concisely present classes of equation systems in Section 3.5.

### 2.5.2 Boolean Equation Systems for Model Checking

Below, we provide the translation of the model checking problem to the problem of solving a Boolean equation system. The transformer $\mathsf{E}$ reduces the global model checking problem $L, \eta \models \phi$ to the problem of solving an equation system.

**Definition 2.35.** Assume $L = \langle S, Act, \rightarrow \rangle$ is a labelled transition system. Let $\phi$ be an arbitrary modal $\mu$-calculus formula over $Act$. Suppose that for every proposition variable $\tilde{X} \in \mathsf{occ}(\phi) \cup \mathsf{bnd}(\phi)$, we have a set of fresh proposition variables $\{X_s \mid s \in S\} \subseteq \mathscr{X}$.

$$
\begin{array}{lcl}
\mathsf{E}^L(b) & \triangleq & \epsilon \\
\mathsf{E}^L(\tilde{X}) & \triangleq & \epsilon \\
\mathsf{E}^L(f \wedge g) & \triangleq & \mathsf{E}^L(f)\ \mathsf{E}^L(g) \\
\mathsf{E}^L(f \vee g) & \triangleq & \mathsf{E}^L(f)\ \mathsf{E}^L(g) \\
\mathsf{E}^L([A]f) & \triangleq & \mathsf{E}^L(f) \\
\mathsf{E}^L(\langle A \rangle f) & \triangleq & \mathsf{E}^L(f) \\
\mathsf{E}^L(\sigma \tilde{X}.\, f) & \triangleq & (\sigma\{(X_s = \mathsf{RHS}_s(f)) \mid s \in S\})\ \mathsf{E}^L(f)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{RHS}_s(b) & \triangleq & b \\
\mathsf{RHS}_s(\tilde{X}) & \triangleq & X_s \\
\mathsf{RHS}_s(f \wedge g) & \triangleq & \mathsf{RHS}_s(f) \wedge \mathsf{RHS}_s(g) \\
\mathsf{RHS}_s(f \vee g) & \triangleq & \mathsf{RHS}_s(f) \vee \mathsf{RHS}_s(g) \\
\mathsf{RHS}_s([A]f) & \triangleq & \bigsqcap\{\mathsf{RHS}_t(f) \mid a \in A, s \xrightarrow{a} t\} \\
\mathsf{RHS}_s(\langle A \rangle f) & \triangleq & \bigsqcup\{\mathsf{RHS}_t(f) \mid a \in A, s \xrightarrow{a} t\} \\
\mathsf{RHS}_s(\sigma \tilde{X}.\, f) & \triangleq & X_s
\end{array}
$$

Observe that the definition of $\mathsf{E}$ provided here coincides semantically with the definition given in [Mad97] for modal $\mu$-calculus formulae $\phi$; the only deviation is a syntactic one, ensuring that the $[\_]$ and $\langle \_ \rangle$ modalities are mapped onto proposition formulae with $\wedge$, and $\vee$ as their main logical connectives in case there is a non-empty set of emanating transitions, by using $\bigsqcap$ and $\bigsqcup$ instead of $\bigwedge$ and $\bigvee$.

The relation between the original local model checking problem and the problem of solving a Boolean equation system is stated by the theorem below.

**Theorem 2.36** ([Mad97])**.** *Assume $L = \langle S, Act, \rightarrow \rangle$ is a labelled transition system. Let $\sigma\tilde{X}.f$ be an arbitrary modal $\mu$-calculus formula, and let $\theta$ be an arbitrary environment. Then:*

$$L, s, \theta \models \sigma\tilde{X}.f \text{ if and only if } (\llbracket \mathsf{E}^L(\sigma\tilde{X}.f) \rrbracket \eta)(X_s) = \mathsf{true}$$

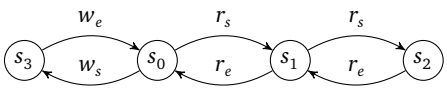
*where for all proposition variables $Y_t$ we set $\eta(Y_t) = \mathsf{true}$ if $t \in \theta(\tilde{Y})$, and* false *for all other proposition variables.*

Informally, the theorem expresses that a state $s$ satisfies a modal $\mu$-calculus formula $\sigma\tilde{X}.f$ if, and only if, the associated proposition variable $X_s$ in the equation system $\mathsf{E}^L(\sigma\tilde{X}.f)$ has true as its solution. The environment $\eta$ ensures that free proposition

variables are correctly dealt with. The correspondence between the global model checking problem and the solution to an equation system then follows immediately from the latter's correspondence to the local model checking problem.

The example below illustrates the above translation and theorem.

**Example 2.37.** Recall the labelled transition system from Example 2.2 (depicted below), modelling mutual exclusion between two readers and a single writer.




The verification problem $\nu\tilde{X}.\mu\tilde{Y}.\ \langle r_s\rangle\tilde{X} \vee \langle \overline{r_s}\rangle\tilde{Y}$, modelling that on some path, a reader can infinitely often start reading, translates to the following equation system using the translation $\mathsf{E}$:

$$\begin{aligned}
&\nu X_{s_0} = Y_{s_0}\\
&\nu X_{s_1} = Y_{s_1}\\
&\nu X_{s_2} = Y_{s_2}\\
&\nu X_{s_3} = Y_{s_3}\\
&\mu Y_{s_0} = (X_{s_1} \vee X_{s_1}) \vee (Y_{s_3} \vee Y_{s_3})\\
&\mu Y_{s_1} = (X_{s_2} \vee X_{s_2}) \vee (Y_{s_0} \vee Y_{s_0})\\
&\mu Y_{s_2} = \mathsf{false} \vee (Y_{s_1} \vee Y_{s_1})\\
&\mu Y_{s_3} = \mathsf{false} \vee (Y_{s_0} \vee Y_{s_0})
\end{aligned}$$

Observe that, like the original $\mu$-calculus formula, which has mutual dependencies between $\tilde{X}$ and $\tilde{Y}$, the resulting equation system has mutual dependencies between the indexed $X$ and $Y$ variables. Solving the resulting equation system leads to $\mathsf{true}$ for all bound variables; $X_{s_i} = \mathsf{true}$, for arbitrary state $s_i$, implies that the property holds in state $s_i$. Furthermore, note that the right-hand sides of the resulting equation system can be rewritten using standard rules of logic, removing, *e.g.*, all occurrences of $\mathsf{false}$. It is not hard to check that this does not affect the solution to the equation system.

## 2.6 Parity Games

Parity games [EJ91; McN93; Zie98] are played by two players, called *even* and *odd*, represented by $\diamond$ and $\Box$. Note that in the literature they are also sometimes referred to as *Abelard* and *Eloise* or as *Duplicator* and *Spoiler*. The game is played on a total, finite directed graph in which vertices have been assigned priorities. Every vertex in the graph belongs to exactly one of these two players. The game is played by moving a token along the edges in the graph indefinitely; the edge that is moved along is chosen by the player owning the vertex on which the token currently resides. Priorities that appear infinitely often along such infinite plays then determine the winner of the play. If the *parity* of the greatest priority that occurs infinitely often is *even* then player $\diamond$ wins the play, otherwise player $\Box$ wins. We say that a vertex is *won* by a player if that player can play the game in such a way that, regardless of the moves made by her opponent, she can win every play through that vertex. In this case, she has a *winning strategy* from that vertex. The

objective of the game is to find the partitioning that separates the vertices won by $\Diamond$ from those won by $\Box$.

By solving a parity game we mean computing the set of vertices that, if the token is initially placed on a vertex in this set, allows player *even* (resp. *odd*) to win. This problem is known to be in NP ∩ co-NP, and even in UP ∩ co-UP [Jur98]; it is still an open problem whether a polynomial time algorithm exists for the problem, but even in case such an algorithm is found, it may not be the most efficient algorithm in practice.

The complexity of parity game solving, and the practical interest in the model checking problem, has resulted in lots of researchers investigating efficient algorithms for solving parity games. In the following let $n$ be the number of vertices in a parity game, $m$ the number of edges, and $d$ the number of priorities.

The classical parity game solving algorithm stems from the proof that parity games are memoryless determined in the form presented by McNaughton [McN93] and Zielonka [Zie98]. Other algorithms are based on strategy improvement, *i.e.*, they assign some initial strategy, and then iteratively improve the strategy until a fixed point is reached. The classical version of this algorithm is due to Vöge and Jurdziński [VJ00], with a variation due to Schewe [Sch08a] with an improved complexity. Randomised versions of strategy improvement have been presented by Petersson and Vorobyov [PV01], Björklund, Sandberg and Vorobyov [BSV03; BV07]. Another algorithm following an iterative approach is the small progress measures algorithm due to Jurdziński [Jur00], which attaches a measure characterising the reachable cycles to every vertex, and iteratively changes that measure. Additionally there are algorithms that combine features of different approaches such as the *bigstep* algorithm due to Schewe [Sch07] which uses small progress measures to compute small winning regions for one of the players, and then uses this as basic step in the recursive algorithm. Jurdziński, Paterson and Zwick use an alternative approach to compute dominions of a guaranteed size [JPZ06], leading to a deterministic subexponential algorithm which runs in $n^{\mathcal{O}(\sqrt{n})}$ time. These algorithms all solve parity games globally. A local algorithm was presented by Stevens and Stirling [SS98]. Apart from the dominion decomposition algorithm from [JPZ06], the algorithms are exponential in the number of priorities $d$ of the parity game, where the best complexity is that of the bigstep algorithm, *viz.* $\mathcal{O}(mn^{d/3})$.

For some of these algorithms, tightness of the worst-case running time has long been an open problem. Upper bounds were established, but it was unknown whether classes of problems exist for which the algorithm indeed behaves exponentially. Friedmann has given classes of parity games for these algorithms that indeed show that all known parity game solving algorithms are exponential [Fri09; Fri10; Fri11b; Fri13; Fri11c; Fri11a; FHZ11]. In practice the recursive algorithm, probably due to its simplicity, is among the best algorithms although it does not have the best worst-case complexity [Kei09; FL09].

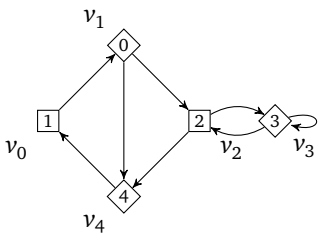
**Definition 2.38.** A parity game $\mathcal{G}$ is a directed graph $(V, \rightarrow, \Omega, \mathcal{P})$, where

- $V$ is a finite set of vertices,
- $\rightarrow \subseteq V \times V$ is a total edge relation (*i.e.*, for each $v \in V$ there is at least one $w \in V$ such that $(v, w) \in \rightarrow$),
- $\Omega : V \rightarrow \mathbb{N}$ is a priority function that assigns priorities to vertices,

- $\mathscr{P} : V \to \{\Diamond, \Box\}$ is a function assigning vertices to players.

Instead of $(v, w) \in \longrightarrow$ we usually write $v \to w$. If $i$ is a player, then $\neg i$ denotes the opponent of $i$, *i.e.*, $\neg\Diamond = \Box$ and $\neg\Box = \Diamond$. Note that, for the purpose of readability later in this thesis, our definition deviates from the conventional one: instead of requiring a partitioning of $V$ into vertices owned by player even and vertices owned by player odd, we achieve the same through the function $\mathscr{P}$.

**Example 2.39.** We depict a parity game in the figure below. Note that we have labelled the vertices $v_0, \ldots, v_4$ purely for notational convenience. We refer back to this example when we introduce concepts in the rest of this section.




In this parity game vertices $v_1$, $v_3$ and $v_4$ are owned by player $\Diamond$, and $v_0$ and $v_2$ are owned by $\Box$, *i.e.*, $\mathscr{P}(v_1) = \mathscr{P}(v_3) = \mathscr{P}(v_4) = \Diamond$ and $\mathscr{P}(v_0) = \mathscr{P}(v_2) = \Box$. Priorities are assigned as follows: $\Omega(v_0) = 1$, $\Omega(v_1) = 0$, $\Omega(v_2) = 2$, $\Omega(v_3) = 3$ and $\Omega(v_4) = 4$. Observe that vertices owned by player $\Diamond$ may be assigned priorities of an odd parity and vice versa.

### 2.6.1 Paths

A sequence of vertices $v_1, \ldots, v_n$ for which $v_i \to v_{i+1}$ for all $1 \leq i < n$ is called a *path*, and may be denoted using angular brackets: $\langle v_1, \ldots, v_n \rangle$. The concatenation $p \cdot q$ of paths $p$ and $q$ is again a path, provided there is a transition from the last vertex in $p$ to the first vertex in $q$. We use $p_n$ to denote the $n^{\text{th}}$ vertex in a path $p$. The set of paths of length $n$, for $n \geq 1$ starting in a vertex $v$ is defined inductively as follows.

$$\Pi^1(v) \triangleq \{\langle v \rangle\}$$

$$\Pi^{n+1}(v) \triangleq \{\langle v_1, \ldots, v_n, v_{n+1} \rangle \mid \langle v_1, \ldots, v_n \rangle \in \Pi^n(v) \wedge v_n \to v_{n+1}\}$$

We use $\Pi^\omega(v)$ to denote the set of infinite paths starting in $v$. The set of all paths starting in $v$, both finite and infinite is defined as follows:

$$\Pi(v) \triangleq \Pi^\omega(v) \cup \bigcup_{n \in \mathbb{N}} \Pi^n(v)$$

We liberally refer to an infinite path as a *play*.

**Example 2.40.** Let $p$ be a path such that there is an edge from its last vertex to its first vertex, then we use $p^\omega$ to denote the infinite repetition of $p$. Now $\langle v_0, v_1, v_2, v_3 \rangle$, $\langle v_2 \rangle \cdot \langle v_3 \rangle^\omega$, $\langle v_0, v_1, v_2, v_3, v_2, v_4 \rangle^\omega$, and $\langle v_0, v_1, v_4 \rangle^\omega$ are examples of paths in the parity game of Example 2.39—the list is not exhaustive. Observe that the first path is finite, and the others are infinite. The second path visits $v_2$ once and $v_3$ infinitely many times, and the last two paths visit all vertices on the path infinitely often.

### 2.6.2 Winner of a Play

A game starting in a vertex $v \in V$ is played by placing a token on $v$, and then moving the token along the edges in the graph. Moves are taken indefinitely according to the following simple rule: if the token is on some vertex $v$, player $\mathscr{P}(v)$ moves the token to some vertex $w$ such that $v \to w$. The result is an infinite path $p$ in the game graph. The *parity* of the greatest priority that occurs infinitely often on $p$ defines the *winner* of the path. If this priority is even, then player $\diamond$ wins, otherwise player $\square$ wins.

Note that in this thesis, by using the greatest priority that occurs infinitely often to decide the winner, we consider *max-parity* games; *min-parity* games are completely dual, and look at the parity of the *least* priority that occurs infinitely often on a path to determine the winner of the path. Converting one into the other is a simple linear transformation.

**Example 2.41.** Consider the infinite paths from Example 2.40 on the parity game from Example 2.39. On the first infinite path the only priority that occurs infinitely often is 3, hence this is won by player $\square$. On the second infinite path the priorities 0, 1, 2, 3 and 4 occur infinitely often. The greatest priority is 4, which is even, hence this play is won by player $\diamond$. The last path contains priorities 0, 1 and 4 infinitely often, hence this is also won by $\diamond$.

### 2.6.3 Strategies

A *strategy* for player $i$ is a partial function $\sigma : V^* \to V$, that for each path ending in a vertex owned by player $i$ determines the next vertex to be played onto. The set of strategies for player $i$ in a game $\mathscr{G}$ is denoted $\mathbb{S}^*_{\mathscr{G},i}$, or simply $\mathbb{S}^*_i$ if $\mathscr{G}$ is clear from the context. If a strategy yields the same vertex $w$ for every pair of paths that end in vertex $v$, then the strategy is said to be *memoryless*. The set of memoryless strategies for player $i$ in a game $\mathscr{G}$ is denoted $\mathbb{S}_{\mathscr{G},i}$, abbreviated to $\mathbb{S}_i$ when $\mathscr{G}$ is clear from the context. A memoryless strategy is usually given as a partial function $\sigma : V \to V$.

A path $p$ of length $n$ is *consistent* with a strategy $\sigma \in \mathbb{S}^*_i$, denoted $\sigma \Vdash p$, if and only if for all $1 \leq j < n$ it is the case that $\langle p_1, \ldots, p_j \rangle \in \mathsf{dom}(\sigma)$ and $\mathscr{P}(p_j) = i$ imply $p_{j+1} = \phi(\langle p_1, \ldots, p_j \rangle)$. The definition of consistency is extended to infinite paths in the obvious manner. We denote the set of paths that are consistent with a given strategy $\sigma$, starting in a vertex $v$ by $\Pi_\sigma(v)$; formally, we define:

$$\Pi_\sigma(v) \stackrel{\Delta}{=} \{p \in \Pi(v) \mid \sigma \Vdash p\}$$

We write $\Pi^n_\sigma(v)$ for paths of length $n$ consistent with $\sigma$.

A strategy $\sigma \in \mathbb{S}^*_i$ is said to be a *winning strategy* from a vertex $v$ if and only if $i$ is the winner of every infinite path consistent with $\sigma$. A vertex is won by $i$ if $i$ has a winning strategy from that vertex.

**Example 2.42.** Recall the parity game from Example 2.39. We only give the strategies from vertices for which there is more than one outgoing edge, the other strategies are defined implicitly.

Consider a strategy such that $\diamond$ plays the token from $v_1$ to $v_4$ if the token has been on $v_1$ an even number of times, and to $v_2$ otherwise, and from $v_3$ to $v_3$ every $10^{th}$ time, and

to $v_2$ all other times. $\Box$ always plays from $v_2$ to $v_4$. Observe that player $\Diamond$ wins all plays consistent with these strategies, and that the strategy for player $\Diamond$ uses memory.

Alternatively consider memoryless strategies in which $\Diamond$ always plays from $v_1$ to $v_4$ and from $v_3$ to $v_2$, and where player $\Box$ always plays from $v_2$ to $v_3$. Observe that every play consistent with these strategies, and starting in vertices $v_0$, $v_1$ or $v_4$ is won by $\Diamond$, and every play consistent with these strategies starting in $v_2$ or $v_3$ is won by player $\Box$. These strategies are optimal, *i.e.*, both player cannot improve upon their strategy such that they can win from additional vertices.

It is well-known that parity games are memoryless determined, *i.e.*, each vertex in the game is won by exactly one player, and if a winning strategy for a player exists from a vertex, then also a winning memoryless strategy exists.

**Theorem 2.43** ([Mar75; GH82; EJ91; McN93])**.** *For every parity game there is a unique partition $(W_\Diamond, W_\Box)$ such that winning strategies $\sigma_\Diamond \in \mathbb{S}^*_\Diamond$ from $W_\Diamond$ and $\sigma_\Box \in \mathbb{S}^*_\Box$ from $W_\Box$ exist. Furthermore, if $\sigma_\Diamond \in \mathbb{S}^*_\Diamond$ is winning from $W_\Diamond$ also a memoryless strategy $\sigma'_\Diamond \in \mathbb{S}_\Diamond$ that is winning from $W_\Diamond$ exists, likewise for $\Box$.*

### 2.6.4 Ordering

We assume that $V$ is ordered by an arbitrary, total ordering $\sqsubset$. The minimal element of a non-empty set $U \subseteq V$ with respect to this ordering is denoted $\sqcap(U)$. Note that this element always exists, since we consider finite parity games. This ordering plays a crucial role in, *e.g.*, proving correctness of stuttering equivalence in Chapter 4, where it is used to construct a strategy that mimics a given strategy.

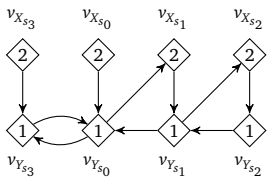
### 2.6.5 Relation to Boolean Equation Systems

There is a one-to-one correspondence between Boolean equation systems in simple recursive form to parity games. In Chapter 3 we investigate graph structures for Boolean equation systems without restrictions on the right hand sides, and we show the consequences of translating a BES to SRF.

**Definition 2.44.** Let $\mathscr{E}$ be a closed Boolean equation system in SRF. We construct a parity game $\mathscr{G}_\mathscr{E} = (V, \rightarrow, \Omega, \mathscr{P})$ as follows.

- $V \triangleq \{v_X \mid X \in \mathsf{bnd}(\mathscr{E})\}$ are the vertices of the game,
- $\rightarrow \triangleq \{v_X \rightarrow v_Y \mid Y \in \mathsf{occ}(\varphi_X)\}$ are the edges of the game,
- ranks are used as priorities $\Omega(v_X) \triangleq \mathsf{rank}_\mathscr{E}(X)$, and
- players are determined using the operator in the right hand side, *i.e.* $\mathscr{P}(v_X) \triangleq \begin{cases} \Box & \text{if } \varphi_X = \bigwedge F, \text{ for some } F \subseteq \mathscr{X} \\ \Diamond & \text{otherwise.} \end{cases}$

**Example 2.45.** Consider the BES from Example 2.37. This is translated to the following parity game.




The following lemma [Mad97; Kei06; Kei09] formalises the correspondence.

**Lemma 2.46.** *Let $\mathscr{E}$ be a Boolean equation system in SRF, then player $\diamond$ wins from $v_X$ in $\mathscr{G}_{\mathscr{E}}$ if and only if $[\![\mathscr{E}]\!](X) = \mathsf{true}$.*

The translation from a parity game to a BES is similar.

**Definition 2.47.** Let $\mathscr{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. The corresponding BES $\mathscr{E}_{\mathscr{G}}$ contains equations $\sigma_{X_v} X_v = \bigwedge \{X_w \mid v \to w\}$ for $v \in V$ such that $\mathscr{P}(v) = \Box$, and $\sigma_{X_v} X_v = \bigvee \{X_w \mid v \to w\}$ for $v \in V$ such that $\mathscr{P}(v) = \diamond$. Furthermore $\sigma_{X_v} = \nu$ if and only if $\Omega(v)$ is even, and if $\Omega(v) > \Omega(w)$, then $\mathsf{rank}_{\mathscr{E}}(X_v) < \mathsf{rank}_{\mathscr{E}}(X_w)$.

**Example 2.48.** The parity game from Example 2.39 is translated to the following BES.

$$\begin{aligned}
\nu X_{v_4} &= X_{v_0} \\
\mu X_{v_3} &= X_{v_3} \vee X_{v_2} \\
\nu X_{v_2} &= X_{v_3} \wedge X_{v_4} \\
\mu X_{v_0} &= X_{v_1} \\
\nu X_{v_1} &= X_{v_4} \vee X_{v_2}
\end{aligned}$$

The correspondence is formalised in the following lemma, which was described in [Mad97; Kei06; Kei09].

**Lemma 2.49.** *Let $\mathscr{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. Let $v \in V$, then $[\![\mathscr{E}_{\mathscr{G}}]\!](X_v) = \mathsf{true}$ if and only if player $\diamond$ wins from $v$.*

# Chapter 3

# Structural Operational Semantics for Boolean Equation Systems

Graph structures and Boolean equation systems have been widely studied as intermediate formats for verification problems such as model checking. In Section 2.5 we have given an overview of these graph structures, and we introduced parity games as a specific instance of such graphs in Section 2.6. In this chapter we limit ourselves to graph structures that represent Boolean equation systems.

The one thing all these graph representations have in common is that they require the right hand sides of the equations in the BES to be either purely conjunctive or purely disjunctive. As such they all closely represent subsets of parity games or, equivalently, Boolean equation systems in SRF. The graphs are also closely related to the games proposed by Stirling (see *e.g.* [Sti97; SS98]), in which players aim to win an infinite game. It has been shown on several occasions that the latter problem is equivalent to solving an equation system. Stirling's game graphs were implemented in various tools, most notably in the Concurrency Workbench.

From a practical viewpoint, the class of equation systems in SRF does not pose any limitations since arbitrary BESs can be translated into SRF in linear-time. Henceforth we refer to this transformation process as *normalising* a BES. In this chapter we study the effects of normalisation in more detail. To this end we first investigate the question.

> Is there a graph structure for Boolean equation systems capturing BESs in their full generality?

Solving a Boolean equation system is expensive, and depends on the size of the equation system. In [KW11] strong bisimulation reduction for equation systems in SRF was studied. In this chapter we investigate whether we can effectively reduce equation systems by reducing their structure graphs using a similar notion of strong bisimulation. We therefore desire that our structure graphs are such that we can:

> Define bisimilarity for structure graphs, and show that BESs obtained from bisimilar structure graphs have the same solution.

The main problem in obtaining our results is that it is hard to elegantly capture the structure of an equation system, without resulting in a parse-tree of the equation system. As a matter of fact, bisimilarity on structure graphs is required to reflect associativity and commutativity of Boolean operators such as $\wedge$ and $\vee$ in order to obtain our second result; this cannot be achieved using parse-trees. In addition, the arbitrary nesting levels of Boolean operators in equation systems complicate a straightforward definition of bisimilarity for such general equation systems. We solve these issues by using a set of deduction rules in Plotkin style [Plo04] to map the equation systems onto *structure graphs*. The latter generalise the dependency graphs for equation systems by dropping the requirement that each vertex necessarily represents a proposition variable occurring at the left-hand side of some equation and adding facilities for reasoning about Boolean constants true and false, and unbound variables.

We use bisimilarity to study the effects of normalisation. Particularly we investigate the effects of normalisation on the minimising capabilities of bisimulation. This leads to the following question:

> What is the effect of normalising an equation system on the minimising capabilities of bisimulation? In other words: how does the size of the *bisimulation quotient* of an equation system compare to the size of the bisimulation quotient of its normalised counterpart?

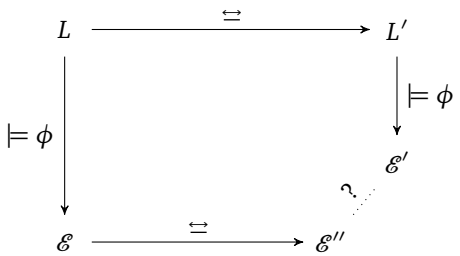
We answer this question in favour of the process of normalisation: the size of the quotient of the normalised equation system will be at most the size of the quotient of the original equation system (see Theorem 3.27). In addition, we provide an example (see Example 3.28) in which the quotient is strictly smaller in size.

In [KW11], a weaker notion of bisimulation, *idempotence identifying bisimulation* was studied for equation systems in SRF. Using this notion, equations such as $(\nu X = X \wedge X)(\nu Y = X)$, in which the right hand sides are equivalent modulo idempotence, can be related. We generalise the definition of idempotence identifying bisimilarity to structure graphs, and we study whether it is indeed true to its name.

> Does idempotence identifying bisimilarity indeed reflect idempotence?

We answer this question negatively, and in answering this question, we show that a meaningful idempotence identifying bisimilarity only exists for a subset of structure graphs.

It is well-known that the modal $\mu$-calculus is preserved under bisimulation minimisation of the behavioural state space, see Theorem 2.16. As the size of the BES encoding a model checking problem is proportional to the size of the state space, minimising the state space prior to verification (by whatever global method) can be a useful pre-computation step, provided that the state space is available. In some methodologies, BESs are however generated from symbolic state spaces, see *e.g.* [GW05a; GW05b]. It is unknown whether state space minimisation and minimisation of equation systems encoding a model checking problem are comparable, in the sense that bisimilar states in an LTS, when combined with the same model checking problem, give rise to bisimilar vertices in the resulting structure graph, see also the picture below.




This naturally leads to the following question:

> Do bisimilar states in a state space give rise to bisimilar equations in the equation systems encoding model checking problems?

The answer to this question is stated by Proposition 3.55, confirming that pairs of bisimilar states in some state space $L$ induce equations in $\mathscr{E}$ that can also be related through an appropriate bisimulation relation underlying the equation system encoding the model checking problem $L \models \phi$. This result remains valid when considering 'safe' abstractions on the original state space. This is shown in Theorem 3.59. We moreover provide an example, see Example 3.61, in which we show that the bisimulation reduction of equation systems can be arbitrarily larger than the reduction of state spaces, even in the presence of safe abstractions.

**Structure of this chapter.** Section 3.1 introduces structure graphs, defines bisimulation for structure graphs, and presents the deduction rules for generating structure graphs from an equation system. In Section 3.2 we give deduction rules for normalisation of equation systems. Bisimulation on structure graphs, its effect on the solution of the underlying equation system, and the interplay with normalisation are studied in Section 3.3. In Section 3.4 we weaken bisimulation to cater for some forms of idempotence. The relation between bisimulation on processes and bisimulation on structure graphs is investigated in Section 3.5. The chapter is concluded with an application of our theory in Section 3.6.

## 3.1 Structure Graphs for Boolean Equation Systems

A large part of the complexity of equation systems is attributed to the mutual dependencies between the equations. These intricate dependencies are neatly captured by *structure graphs*.

A structure graph is defined in the context of a set of proposition variables $\mathscr{X}$, which generally contains at least the bound and occurring proposition variables in an equation system.

**Definition 3.1.** Given a set of proposition variables $\mathscr{X}$. A structure graph over $\mathscr{X}$ is a vertex-labelled graph $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$, where:

- $T$ is a finite set of vertices;

- $t \in T$ is the initial vertex;
- $\rightarrow \subseteq T \times T$ is a dependency relation, and

partial functions $d, r$ and $\nearrow$ are such that

- $d : T \rightarrow \{\blacktriangle, \blacktriangledown, \top, \bot\}$ is a vertex decoration mapping;
- $r : T \rightarrow \mathbb{N}$ is a vertex ranking mapping;
- $\nearrow : T \rightarrow \mathscr{X}$ is a free variable mapping.

We refer to the domain of partial functions $d$, $r$ and $\nearrow$ using $\mathsf{dom}(d)$, $\mathsf{dom}(r)$ and $\mathsf{dom}(\nearrow)$, respectively.

A structure graph can be used to capture the dependencies between bound variables and (sub)formulae occurring in the equations of such bound variables. Intuitively, the decoration mapping $d$ reflects whether the top symbol of a proposition formula is $\mathsf{true}$ (represented by $\top$), $\mathsf{false}$ (represented by $\bot$), a conjunction (represented by $\blacktriangle$), or a disjunction (represented by $\blacktriangledown$). The vertex ranking mapping $r$ indicates the rank of a vertex. The free variable mapping indicates whether a vertex represents a free variable. Note that each vertex can have at most one rank, at most one decoration $\star \in \{\blacktriangle, \blacktriangledown, \top, \bot\}$, and at most one free variable $\nearrow_X$. We sometimes write $t$ to refer to a structure graph $\langle T, t, \rightarrow, d, r, \nearrow \rangle$, where $t$ is in fact the initial vertex of the structure graph.

We define the size of a structure graph $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ as $|\mathscr{G}| = |T|$, *i.e.*, the size of a structure graph is the number of vertices in the graph.

In Section 3.1.1, we show how a structure graph can be obtained systematically from a formula and an equation system. In Section 3.1.2, we present the reverse, *i.e.*, we define how an equation system can be associated with a structure graph assuming that it satisfies some well-formedness constraints.

One can easily define strong bisimilarity on structure graphs using the standard notion of bisimulation, and requiring that the decorations are the same for related vertices.

**Definition 3.2.** Let $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be a structure graph. A symmetric relation $R \subseteq T \times T$ is a strong bisimulation relation if for all $(u, u') \in R$

- $d(u) = d(u')$, $r(u) = r(u')$, and $\nearrow(u) = \nearrow(u')$;
- for all $v \in T$, if $u \rightarrow v$, then $u' \rightarrow' v'$ for some $v' \in T$ such that $(v, v') \in R$.

Two vertices $u$ and $u'$ are bisimilar, notation $u \leftrightarrows u'$ if and only if there exists a strong bisimulation relation $R$ such that $(u, u') \in R$.

Note that in the above definition, we abuse notation to require that $r(u) = r(u')$ also if $u, u' \notin \mathsf{dom}(r)$, likewise for $d$ and $\nearrow$. Strong bisimulation equivalence on structure graphs is an equivalence relation.

**Proposition 3.3.** *$\leftrightarrows$ is an equivalence relation.*

Using the notion of bisimilarity, we also define the *strong bisimulation quotient* of a structure graph.

**Definition 3.4.** Let $\mathcal{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be a structure graph. The bisimulation quotient $\mathcal{G}_{/\leftrightarrows} = \langle T', t', \rightarrow', d', r', \nearrow' \rangle$ of $\mathcal{G}$ is defined as follows:

- $T' \triangleq T_{/\leftrightarrows} = \{[t_i]_{\leftrightarrows} \mid t_i \in T\}$ with $[t_i]_{\leftrightarrows} = \{t_j \in T \mid t_i \leftrightarrows t_j\}$;
- $t' \triangleq [t]_{\leftrightarrows}$;
- $\rightarrow' \triangleq \{[t_i]_{\leftrightarrows} \rightarrow' [t_j]_{\leftrightarrows} \mid t_i \rightarrow t_j\}$
- $d'([t_i]_{\leftrightarrows}) \triangleq d(t_i)$, if $t_i \in \mathsf{dom}(d)$, and undefined otherwise;
- $r'([t_i]_{\leftrightarrows}) \triangleq r(t_i)$, if $t_i \in \mathsf{dom}(r)$, and undefined otherwise;
- $\nearrow' ([t_i]_{\leftrightarrows}) \triangleq \nearrow (t_i)$, if $t_i \in \mathsf{dom}(\nearrow)$, and undefined otherwise.

The decorations in the quotient are uniquely defined because of the first requirement in the definition of bisimulation.

### 3.1.1 Structural Operational Semantics for Equation Systems

Next, we define structure graphs $\langle f, \mathcal{E} \rangle$ generated by an arbitrary equation system $\mathcal{E}$ and proposition formula $f$. In defining our structure graphs, we use Plotkin-style Structural Operational Semantics [Plo04] to associate a structure graph to a formula $f$ in the context of an equation system $\mathcal{E}$, notation $\langle f, \mathcal{E} \rangle$. The deduction rules define a relation $\_ \rightarrow \_$ and predicates $\_ \Darr n$ (for $n \in \mathbb{N}$), $\_ \nearrow_X$ (for $X \in \mathcal{X}$), $\_\top$, $\_\bot$, $\_\blacktriangle$, and $\_\blacktriangledown$. The rules are presented in the form $\frac{P}{C}$ where $P$ describes the premise, and $C$ the conclusion. Intuitively, we can generate $C$ if $P$ holds. In the deduction rules also negative premises are used. When we use negative premises, we need to take care that we do this in such a way that the rules are still well-defined. See [MRG05] for an overview of negative premises in structural operational semantics.

We would like the structure graphs to be such that bisimilarity is able to relate logically equivalent vertices. To simplify this, we restrict ourselves to detecting some simple logical equivalences. Bisimilarity of structure graphs should at least reflect symmetry and associativity of the proposition formulae, *i.e.*, the structure graphs of formulae $f \wedge g$ and $g \wedge f$ in the context of equation system $\mathcal{E}$ should be bisimilar. Likewise, structure graphs for $f \wedge (g \wedge h)$ and $(f \wedge g) \wedge h$ in the context of $\mathcal{E}$ should be bisimilar. Intuitively, for equation systems in SRF this choice allows us to derive a structure graph that is very much like the parity game that corresponds to the equation system, and we do not introduce any unnecessary complexity.

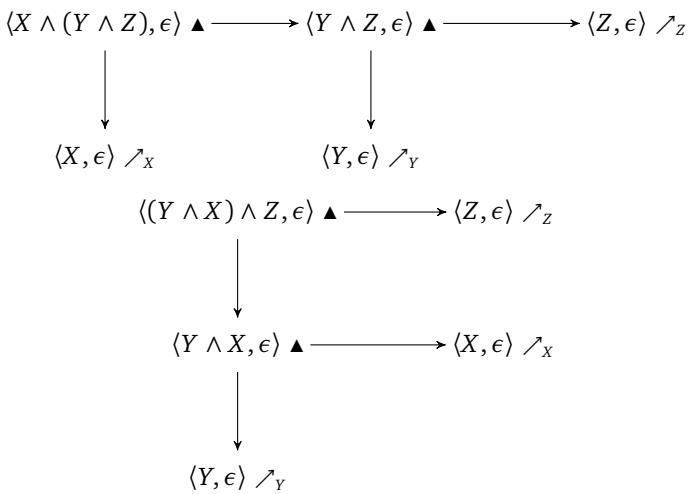
The notations used in the deduction rules deviate slightly from those used in the structure graphs to provide a more graphical notation. The predicate $t \nearrow_X$ represents $\nearrow (t) = X$, the predicate $t \Darr n$ represents $r(t) = n$, for $\star \in \{\blacktriangle, \blacktriangledown, \top, \bot\}$, $t\star$ represents $d(t) = \star$. The notation $t \not\Darr$ represents $\neg(t \Darr n)$ for all $n \in \mathbb{N}$.

The basic building blocks of formulae in an equation system are the proposition variables. Since we are dealing with possibly open equation systems, free variables may occur. We label nodes representing free variables occurring in an equation system as such using

$\nearrow$ in rule 1 below. Furthermore, the rank of bound proposition variables is relevant to the solution of the BES. We use this rank to label vertices representing such bound proposition variables in rule 2.

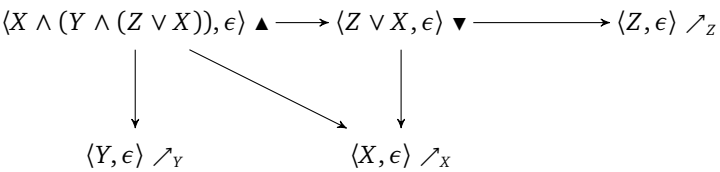
$$(1)\ \frac{X \in \mathsf{occ}(\mathscr{E}) \setminus \mathsf{bnd}(\mathscr{E})}{\langle X, \mathscr{E}\rangle \nearrow_X} \qquad (2)\ \frac{X \in \mathsf{bnd}(\mathscr{E})}{\langle X, \mathscr{E}\rangle \Uparrow \mathsf{rank}_{\mathscr{E}}(X)}$$

In Boolean equation systems, the right-hand sides are built up of binary conjunctions and disjunctions. The next question that needs to be answered is 'How to capture this structure in the structure graph?' One way of doing so would be to precisely reflect the structure of the proposition formula. For a formula of the form $X \wedge (Y \wedge Z)$ in the context of an empty equation system this results in the first structure graph depicted below:




A drawback of this solution is that, in general, the logical equivalence between $X \wedge (Y \wedge Z)$ and $(Y \wedge X) \wedge Z$ (see the second structure graph above) is not reflected by bisimilarity. As discussed before, retaining this logical equivalence, and hence associativity and commutativity of both conjunction and disjunction is desirable. We therefore present a set of deduction rules that uses the context in which conjunctions and disjunctions occur to decide whether intermediate vertices should be introduced or not.

The main difficulty is that the logical connectives for conjunction ($\wedge$) and disjunction ($\vee$) may occur nested in a formula. A change in leading operator in the formula is reflected by introducing an intermediate vertex in the structure graph. As a consequence the anticipated structure of the structure graph for $X \wedge (Y \wedge (Z \vee X))$, in the context of the empty equation system, is:



We next formalise the dependency relations in structure graphs. This can be elegantly achieved by means of the following deduction rules for the decorations and the dependency relation $\rightarrow$:

$$(3)\ \frac{}{\langle \mathsf{true}, \mathscr{E}\rangle \top} \qquad (4)\ \frac{}{\langle \mathsf{false}, \mathscr{E}\rangle \bot} \qquad (5)\ \frac{}{\langle f \wedge f', \mathscr{E}\rangle \blacktriangle} \qquad (6)\ \frac{}{\langle f \vee f', \mathscr{E}\rangle \blacktriangledown}$$

$$(7)\ \frac{\langle f, \mathscr{E}\rangle \blacktriangle \quad \langle f, \mathscr{E}\rangle \not\pitchfork \quad \langle f, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}$$

$$(8)\ \frac{\langle f', \mathscr{E}\rangle \blacktriangle \quad \langle f', \mathscr{E}\rangle \not\pitchfork \quad \langle f', \mathscr{E}\rangle \rightarrow \langle g', \mathscr{E}\rangle}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle g', \mathscr{E}\rangle}$$

$$(9)\ \frac{\langle f, \mathscr{E}\rangle \blacktriangledown \quad \langle f, \mathscr{E}\rangle \not\pitchfork \quad \langle f, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}$$

$$(10)\ \frac{\langle f', \mathscr{E}\rangle \blacktriangledown \quad \langle f', \mathscr{E}\rangle \not\pitchfork \quad \langle f', \mathscr{E}\rangle \rightarrow \langle g', \mathscr{E}\rangle}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle g', \mathscr{E}\rangle}$$

Rules (3-6) describe the axioms for decorations. The deduction rules (7-10) for $\rightarrow$ are introduced to flatten the nesting hierarchy of the same connective, *i.e.*, they are designed to take care of associativity of $\wedge$ and $\vee$. They can be used to deduce that $X \wedge (Y \wedge Z) \rightarrow Y$. Commutativity of $\wedge$ and $\vee$ is implicit because the structure graph does not impose an order on the edges.

Deduction rules 7-10 work for the situation that the subformula has a $\blacktriangle$ or $\blacktriangledown$, but the subformula itself is not a recursion variable (see the second premise of the deduction rules in combination with deduction rules 19 and 20).

$$(11)\ \frac{\neg\langle f, \mathscr{E}\rangle \blacktriangle}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle} \qquad (12)\ \frac{\neg\langle f', \mathscr{E}\rangle \blacktriangle}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle f', \mathscr{E}\rangle}$$

$$(13)\ \frac{\neg\langle f, \mathscr{E}\rangle \blacktriangledown}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle} \qquad (14)\ \frac{\neg\langle f', \mathscr{E}\rangle \blacktriangledown}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle f', \mathscr{E}\rangle}$$

$$(15)\ \frac{\langle f, \mathscr{E}\rangle \pitchfork n}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle} \qquad (16)\ \frac{\langle f', \mathscr{E}\rangle \pitchfork n}{\langle f \wedge f', \mathscr{E}\rangle \rightarrow \langle f', \mathscr{E}\rangle}$$

$$(17)\ \frac{\langle f, \mathscr{E}\rangle \pitchfork n}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle} \qquad (18)\ \frac{\langle f', \mathscr{E}\rangle \pitchfork n}{\langle f \vee f', \mathscr{E}\rangle \rightarrow \langle f', \mathscr{E}\rangle}$$

Deduction rules 11-18 describe the dependencies in case there is no flattening possible anymore (by absence of structure). The deduction rules 11-14 deal with the case that a subformula has no $\blacktriangle$ or $\blacktriangledown$. The deduction rules 15-18 deal with the case that the subformula represents a bound variable.

Finally, we present deduction rules that describe how the structure of a vertex representing a variable is derived from the right-hand side of the corresponding equation. Observe that the deduction rules only have to deal with the case that a defining equation for the recursion variable $X$ has been found in the Boolean equation system. Deduction rules 19 and 20 define the predicates $\blacktriangle$ and $\blacktriangledown$ for the case that the right-hand side is a variable or a constant. Deduction rules 21 and 22 define the dependency relation $\rightarrow$ for the case that the right-hand side is a variable or a constant. Deduction rules 23 and 24 do this for the cases in which the right-hand side is a proposition formula that is neither a variable nor a constant.

$$(19)\ \frac{\sigma X = f \in \mathscr{E} \quad \langle f, \mathscr{E}\rangle\blacktriangle \quad \langle f, \mathscr{E}\rangle \not\Cap}{\langle X, \mathscr{E}\rangle\blacktriangle} \qquad (20)\ \frac{\sigma X = f \in \mathscr{E} \quad \langle f, \mathscr{E}\rangle\blacktriangledown \quad \langle f, \mathscr{E}\rangle \not\Cap}{\langle X, \mathscr{E}\rangle\blacktriangledown}$$

$$(21)\ \frac{\sigma X = f \in \mathscr{E} \quad \neg\langle f, \mathscr{E}\rangle\blacktriangledown \quad \neg\langle f, \mathscr{E}\rangle\blacktriangle}{\langle X, \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle} \qquad (22)\ \frac{\sigma X = f \in \mathscr{E} \quad \langle f, \mathscr{E}\rangle \Cap n}{\langle X, \mathscr{E}\rangle \rightarrow \langle f, \mathscr{E}\rangle}$$

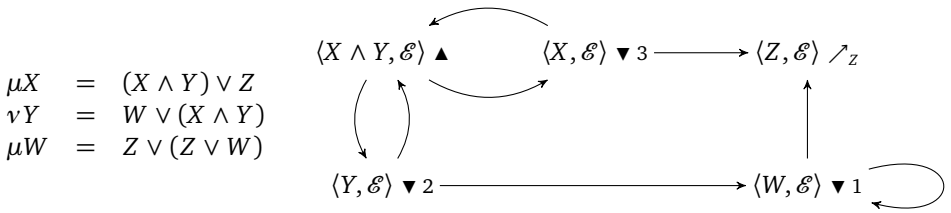
$$(23)\ \frac{\sigma X = f \in \mathscr{E} \quad \langle f, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle \quad \langle f, \mathscr{E}\rangle\blacktriangle \quad \langle f, \mathscr{E}\rangle \not\Cap}{\langle X, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}$$

$$(24)\ \frac{\sigma X = f \in \mathscr{E} \quad \langle f, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle \quad \langle f, \mathscr{E}\rangle\blacktriangledown \quad \langle f, \mathscr{E}\rangle \not\Cap}{\langle X, \mathscr{E}\rangle \rightarrow \langle g, \mathscr{E}\rangle}$$

Since we are using negative premises, we need to provide a stratification, *i.e.*, a mapping from transitions and predicates to ordinals such that for any closed instance of every deduction rule the positive premises are not larger than the conclusion and (the positive instances of) the negative premises are strictly smaller than the conclusion. This ensures that the SOS uniquely defines a collection of transition relations and predicates. In this case, providing such a stratification is easy. As long as the weights of all transitions are larger than the weights of all predicates and the weights of the predicates $\blacktriangle$ and $\blacktriangledown$ are larger than the weights of $\Cap$ predicates, the SOS is stratified. This immediately results in the following proposition.

**Proposition 3.5.** *The set of operational rules to obtain structure graphs from equation systems is meaningful.*

**Example 3.6.** An equation system $\mathscr{E}$ (see left) and its associated structure graph (see right). Observe that the term $X \wedge Y$ is shared by the equations for $X$ and $Y$, and appears only once in the structure graph as an unranked vertex. There is no equation for $Z$; this is represented by the term $Z$, decorated only by the label $\nearrow_Z$. The subterm $Z \vee W$ in the equation for $W$ does not appear as a separate vertex in the structure graph, since the disjunctive subterm occurs within the scope of a disjunction at the top level.




Given a formula $f$ and an equation system $\mathscr{E}$, $\langle f, \mathscr{E}\rangle$ denotes the part of the structure graph generated by the deduction rules that is reachable from the vertex $\langle f, \mathscr{E}\rangle$.

### 3.1.2 Translating Structure Graphs to Equation Systems

Next, we show how, under some mild conditions, a formula and equation system can be obtained from a structure graph. Later in this chapter, we show that, when a structure graph is constructed from a formula in the context of an equation system, and the corresponding vertex is transformed to a new formula with an equation system, then both formulae have the same solution in their respective equation systems.

A structure graph $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow\rangle$ is called *BESsy* if it satisfies the following constraints:

- a vertex $t$ decorated by $\top, \bot$ or $\nearrow_X$ for some $X$ has no successor w.r.t. $\rightarrow$.
- a vertex is decorated by $\blacktriangle$ or $\blacktriangledown$ or a rank if and only if it has at least one successor w.r.t. $\rightarrow$.
- a vertex with multiple successors w.r.t. $\rightarrow$, is decorated with $\blacktriangle$ or $\blacktriangledown$.
- every cycle contains a vertex with a rank.

The following lemma states that any structure graph obtained from a formula and an equation system is BESsy.

**Lemma 3.7.** *For any formula $f$ and equation system $\mathscr{E}$, the structure graph $\langle f, \mathscr{E}\rangle$ is BESsy.*

*Proof.* We have to establish that the structure graph $\langle f, \mathscr{E}\rangle$ is BESsy. Thereto it has to be shown that the four requirements of the definition of BESsyness are satisfied.

The first one trivially follows by considering all the possibilities for generating a vertex labelled by either $\top$, $\bot$, or $\nearrow_X$. In each case it turns out that $f$ is of a form that does not allow the derivation of a $\rightarrow$-transition.

The proof of the second requirement requires induction on the depth of the proof of $\langle f, \mathscr{E}\rangle\blacktriangle$, $\langle f, \mathscr{E}\rangle\blacktriangledown$, or $\langle f, \mathscr{E}\rangle\Cap$, respectively. Inside this induction there is a case distinction on the deduction rule that has been applied last in the proof.

For the proof of the third requirement it suffices to consider all possibilities for generating multiple successors and it follows easily that in these cases the vertex is also labelled by $\blacktriangle$ or $\blacktriangledown$.

The last requirement follows trivially from the observation that a cycle of successor relations can never be generated without using a bound variable along the cycle. This would inevitably introduce a rank for that vertex. □

To show that our notion of structure graph is meaningful, we next present functions that, given a BESsy structure graph, produce a proposition formula and an equation system corresponding to the reachable part of the structure graph. In Section 3.3.2 we use this same notion to show that for bisimilar BESsy structure graphs, the equation systems they induce have the same solution.

For a BESsy structure graph $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ the function $\varphi$ is defined as follows: for $u \in T$

$$\varphi(u) \triangleq \begin{cases} \bigsqcap\{\varphi(u') \mid u \rightarrow u'\} & \text{if } d(u) = \blacktriangle \text{ and } u \notin \mathsf{dom}(r), \\ \bigsqcup\{\varphi(u') \mid u \rightarrow u'\} & \text{if } d(u) = \blacktriangledown \text{ and } u \notin \mathsf{dom}(r), \\ \mathsf{true} & \text{if } d(u) = \top, \\ \mathsf{false} & \text{if } d(u) = \bot, \\ X & \text{if } \nearrow(u) = X, \\ X_u & \text{otherwise.} \end{cases}$$

The function $\varphi$ introduces variables for those vertices that are in the domain of the vertex rank mapping or the free variable mapping. In the second case, the associated variable name is used. In the former case, a fresh variable name is introduced to represent the vertex. For other vertices the structure that is offered via vertex decoration mapping $d$ is used to obtain a formula representing such a structure. In particular, for vertices decorated with $\blacktriangle$ (resp. $\blacktriangledown$), the use of $\bigsqcap$ (resp. $\bigsqcup$) ensures that the resulting formula is conjunctive (resp. disjunctive) by duplicating conjuncts (resp. disjuncts).

**Definition 3.8.** Let $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be a BESsy structure graph. The equation system associated to $\mathscr{G}$, denoted $\beta(\mathscr{G})$, is defined below.

To each vertex $u \in T \cap \mathsf{dom}(r)$, we associate an equation of the form:

$$\sigma X_u = \mathsf{rhs}(u)$$

Here $\sigma$ is $\mu$ in case the rank associated to the vertex is odd, and $\nu$ otherwise. The formula $\mathsf{rhs}(u)$ is defined as follows:

$$\mathsf{rhs}(u) \triangleq \begin{cases} \bigsqcap\{\varphi(u') \mid u \rightarrow u'\} & \text{if } d(u) = \blacktriangle \\ \bigsqcup\{\varphi(u') \mid u \rightarrow u'\} & \text{if } d(u) = \blacktriangledown \\ \varphi(u') & \text{if } d(u) \neq \blacktriangle, d(u) \neq \blacktriangledown, \text{ and } u \rightarrow u' \end{cases}$$

The equation system $\beta(\mathscr{G})$ is obtained by ordering the equations from left-to-right ensuring the ranks of the vertices associated to the equations are descending.

The definition of $\mathsf{rhs}$ and $\varphi$ are closely related. This is formalised by the following lemma.

**Lemma 3.9.** *Let $\mathscr{E}$ be a BES, $(\sigma X = f) \in \mathscr{E}$. Then it holds that $\varphi(\langle f, \mathscr{E} \rangle) = \mathsf{rhs}(\langle X, \mathscr{E} \rangle)$.*

*Proof.* Assume that $(\sigma X = f) \in \mathscr{E}$. Observe that $\langle X, \mathscr{E} \rangle \in \mathsf{dom}(r)$. We proceed by distinguishing the cases in the definition of $\mathsf{rhs}$.

- $d(\langle X, \mathscr{E}\rangle) = \blacktriangle$. Then according to rule (19) also $d(\langle f, \mathscr{E}\rangle) = \blacktriangle$, and furthermore $\langle f, \mathscr{E}\rangle \notin \mathsf{dom}(r)$. We derive:

$$
\begin{array}{ll}
 & \mathsf{rhs}(\langle X, \mathscr{E}\rangle) \\
= & \{\text{Definition of } \mathsf{rhs}\} \\
 & \bigsqcap\{\varphi(u') \mid \langle X, \mathscr{E}\rangle \to u'\} \\
= & \left\{ \begin{array}{l} d(\langle f, \mathscr{E}\rangle) = \blacktriangle \text{ and } \langle X, \mathscr{E}\rangle \notin \mathsf{dom}(r), \text{ hence } \langle X, \mathscr{E}\rangle \to u' \\ \text{if and only if } \langle f, \mathscr{E}\rangle \to u' \text{ according to rule (23)} \end{array} \right\} \\
 & \bigsqcap\{\varphi(u') \mid \langle f, \mathscr{E}\rangle \to u'\} \\
= & \{\text{Definition of } \varphi\} \\
 & \varphi(\langle f, \mathscr{E}\rangle)
\end{array}
$$

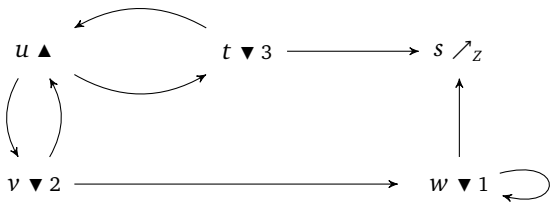
- $d(\langle X, \mathscr{E}\rangle) = \blacktriangledown$. Analogous to the previous case.

- $d(\langle X, \mathscr{E}\rangle) \neq \blacktriangle$ and $d(\langle X, \mathscr{E}\rangle) \neq \blacktriangledown$. We know that there is exactly one $u'$ such that $\langle X, \mathscr{E}\rangle \to u'$, hence using rule (21) we find $\langle X, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$. By definition of $\mathsf{rhs}$, $\mathsf{rhs}(\langle X, \mathscr{E}\rangle) = \varphi(\langle f, \mathscr{E}\rangle)$.

□

We illustrate the use of the functions $\varphi$ and $\beta$ for obtaining a formula and a BES from a structure graph in the following example.

**Example 3.10.** Consider the structure graph below.




In this structure graph we find, *e.g.*, that $\varphi(t) = X_t, \varphi(v) = X_v$, and the more complex $\varphi(u) = \bigsqcap\{\varphi(v), \varphi(t)\} = X_v \wedge (X_t \wedge X_t)$. The corresponding equation system, obtained using $\beta$ is

$$
\begin{array}{lcl}
\mu X_t & = & (X_t \wedge (X_v \wedge X_v)) \vee (Z \vee Z) \\
\nu X_v & = & (X_t \wedge (X_v \wedge X_v)) \vee (X_w \vee X_w) \\
\mu X_w & = & X_w \vee (Z \vee Z)
\end{array}
$$

Observe that the BES that we obtained in the previous example is equivalent to the BES from Example 3.6, modulo renaming of bound proposition variables, and simplification of the right hand sides. This should not be surprising, since the structure graph also was the same, except for renaming of the vertices. We next formalise this correspondence between a BES and the BES obtained from its structure graph.

We show this in two stages. Given a BES $\mathscr{E}$ we show the correspondence between the right hand side of an equation in $\mathscr{E}$, and the right hand side obtained from the structure graph of $\mathscr{E}$. We then generalise this to equation systems.

First we show that the function $\varphi$ distributes over the Boolean connectives when we consider the semantics.

**Lemma 3.11.** *Let $f, g$ be formulae, $\mathscr{E}$ a BES, and $\eta$ an arbitrary environment, then we have the following semantic equivalences:*

$$\begin{aligned}[\![\varphi(\langle f, \mathscr{E}\rangle) \wedge \varphi(\langle g, \mathscr{E}\rangle)]\!]\eta &= [\![\varphi(\langle f \wedge g, \mathscr{E}\rangle)]\!]\eta \\ [\![\varphi(\langle f, \mathscr{E}\rangle) \vee \varphi(\langle g, \mathscr{E}\rangle)]\!]\eta &= [\![\varphi(\langle f \vee g, \mathscr{E}\rangle)]\!]\eta\end{aligned}$$

*Proof.* We prove the first statement. Proof of the second statement is completely symmetric. We prove that each conjunct in the formula on the left-hand side is also a conjunct of the formula on the right hand side, and vice versa. We interpret the $=$ in our goal as a bi-implication, and we split the proof obligation in two implications.

We first prove that $[\![\varphi(\langle f \wedge g, \mathscr{E}\rangle)]\!]\eta \implies [\![\varphi(\langle f, \mathscr{E}\rangle) \wedge \varphi(\langle g, \mathscr{E}\rangle)]\!]\eta$ by induction on the structure of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$.

- $\varphi(\langle f \wedge g, \mathscr{E}\rangle) = \bigsqcap\{\varphi(u') \mid \langle f \wedge g, \mathscr{E}\rangle \to u'\}$. It follows that $d(\langle f \wedge g, \mathscr{E}\rangle) = \blacktriangle$ and $\langle f \wedge g, \mathscr{E}\rangle \notin \mathsf{dom}(r)$ from the definition of $\varphi$. As $d(\langle f \wedge g, \mathscr{E}\rangle) = \blacktriangle$ and $\langle f \wedge g, \mathscr{E}\rangle$ is BESsy, there must be at least one $u'$ such that $\langle f \wedge g, \mathscr{E}\rangle \to u'$. We need to show that for each conjunct $u' \in \{\varphi(u'') \mid \langle f \wedge g, \mathscr{E}\rangle \to u''\}$ either:
  - $u' \in \{\varphi(u'') \mid \langle f, \mathscr{E}\rangle \to u''\}$, or
  - $u' \in \{\varphi(u'') \mid \langle g, \mathscr{E}\rangle \to u''\}$, or
  - $u' = \varphi(\langle f, \mathscr{E}\rangle)$, or
  - $u' = \varphi(\langle g, \mathscr{E}\rangle)$.

  Let $v = \varphi(u')$ be an arbitrary conjunct in $\{\varphi(u'') \mid \langle f \wedge g, \mathscr{E}\rangle \to u''\}$. So we know $\langle f \wedge g, \mathscr{E}\rangle \to u'$. We apply case distinction on the inference rules that can introduce this edge.
  - $\langle f \wedge g, \mathscr{E}\rangle \to u'$ is introduced through rule (7). Then we may assume that $d(\langle f, \mathscr{E}\rangle) = \blacktriangle$, $\langle f, \mathscr{E}\rangle \notin \mathsf{dom}(r)$ and $\langle f, \mathscr{E}\rangle \to u'$. According to the definition of $\varphi$ we find that $\varphi(\langle f, \mathscr{E}\rangle) = \bigsqcap\{\varphi(u'') \mid \langle f, \mathscr{E}\rangle \to u''\}$. Hence by induction we find that $v$ is a conjunct of $\varphi(\langle f, \mathscr{E}\rangle)$. As $d(\langle f, \mathscr{E}\rangle) = \blacktriangle$, every conjunct of this formula is also a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$.
  - $\langle f \wedge g, \mathscr{E}\rangle \to u'$ is introduced through rule (8). This case is analogous to the previous case.
  - $\langle f \wedge g, \mathscr{E}\rangle \to u'$ is introduced through rule (11). We may assume that $\neg\langle f, \mathscr{E}\rangle\blacktriangle$. Therefore, $u' = \langle f, \mathscr{E}\rangle$, and the corresponding formula is $\varphi(\langle f, \mathscr{E}\rangle)$.
  - The cases where $\langle f \wedge g, \mathscr{E}\rangle \to u'$ is introduced through rules (12), (15) or (16) are analogous to the previous case.
- $\varphi(\langle f \wedge g, \mathscr{E}\rangle) = \bigsqcup\{\varphi(u') \mid \langle f \wedge g, \mathscr{E}\rangle \to u'\}$. According to rule (5) it must be the case that $\langle f \wedge g, \mathscr{E}\rangle\blacktriangle$. According to BESsyness then $d(\langle f \wedge g, \mathscr{E}\rangle) \neq \blacktriangledown$, hence $\varphi(\langle f \wedge g, \mathscr{E}\rangle) \neq \bigsqcup\{\varphi(u') \mid \langle f \wedge g, \mathscr{E}\rangle \to u'\}$, hence this case cannot apply.
- the cases where $\varphi(\langle f \wedge g, \mathscr{E}\rangle) \in \{\mathsf{true}, \mathsf{false}, X\}$ are analogous to the previous case.
- $\varphi(\langle f \wedge g, \mathscr{E}\rangle) = X_{\langle f \wedge g, \mathscr{E}\rangle}$. Appealing to rule (5) it must be the case that $\varphi(\langle f \wedge g, \mathscr{E}\rangle)\blacktriangle$. Furthermore we know $\langle f \wedge g, \mathscr{E}\rangle \in \mathsf{dom}(r)$. According to the operational rules all ranked terms are of the form $\langle Y, \mathscr{E}\rangle$, for some $Y$. This contradicts the assumption that the term we are considering is $\langle f \wedge g, \mathscr{E}\rangle$.

The reverse case, showing that $[\![\varphi(\langle f \wedge g, \mathscr{E}\rangle)]\!]\eta \Leftarrow [\![\varphi(\langle f, \mathscr{E}\rangle) \wedge \varphi(\langle g, \mathscr{E}\rangle)]\!]\eta$ commences by a case distinction on the structure of $\varphi(\langle f, \mathscr{E}\rangle)$ and $\varphi(\langle g, \mathscr{E}\rangle)$. We show that each conjunct of $\varphi(\langle f, \mathscr{E}\rangle)$ is also a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$. The proof that every conjunct of $\varphi(\langle g, \mathscr{E}\rangle)$ is a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$ if completely analogous, and therefore omitted.

- case $\varphi(\langle f, \mathscr{E}\rangle) = \bigsqcap\{\varphi(u') \mid \langle f, \mathscr{E}\rangle \to u'\}$. In this case we know that $d(\langle f, \mathscr{E}\rangle) = \blacktriangle$, and $\langle f, \mathscr{E}\rangle \notin \mathsf{dom}(r)$. Let $\langle f, \mathscr{E}\rangle \to u'$, so $\varphi(u')$ is a top level conjunct of $\varphi(\langle f, \mathscr{E}\rangle)$. From rule (7) it follows immediately that $\langle f \wedge g, \mathscr{E}\rangle \to u'$, and $d(\langle f \wedge g, \mathscr{E}\rangle) = \blacktriangle$ according to (5), hence $\varphi(\langle f \wedge g, \mathscr{E}\rangle) = \bigsqcap\{\varphi(u') \mid \langle f \wedge g, \mathscr{E}\rangle \to u'\}$, and $\varphi(u')$ is a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$.

- $\varphi(\langle f, \mathscr{E}\rangle) = \bigsqcup\{\varphi(u') \mid \langle f, \mathscr{E}\rangle \to u'\}$. So we know that $d(\langle f, \mathscr{E}\rangle) = \blacktriangledown$ and $\langle f, \mathscr{E}\rangle \notin \mathsf{dom}(r)$. Observe that the only conjunct of $\varphi(\langle f, \mathscr{E}\rangle)$ is $\varphi(\langle f, \mathscr{E}\rangle)$ itself. We show that $\varphi(\langle f, \mathscr{E}\rangle)$ is a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$. According to rule (11), $\langle f \wedge g, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$. Furthermore $d(\langle f \wedge g, \mathscr{E}\rangle) = \blacktriangle$ according to (5) and $\langle f \wedge g, \mathscr{E}\rangle \notin \mathsf{dom}(r)$ according to (2), hence $\varphi(\langle f \wedge g, \mathscr{E}\rangle) = \bigsqcap\{\varphi(u') \mid \langle f \wedge g, \mathscr{E}\rangle \to u'\}$, and $\varphi(\langle f, \mathscr{E}\rangle)$ is a conjunct of $\varphi(\langle f \wedge g, \mathscr{E}\rangle)$.

- cases $\varphi(\langle f, \mathscr{E}\rangle) \in \{\mathsf{true}, \mathsf{false}, X\}$ follow a similar line of reasoning as the previous case.

- $\varphi(\langle f, \mathscr{E}\rangle) = X_{\langle f, \mathscr{E}\rangle}$, where $\langle f, \mathscr{E}\rangle \in \mathsf{dom}(r)$. This again follows a similar line of reasoning. We use the observation that the only edge that is generated from $\langle f \wedge g, \mathscr{E}\rangle$ induced by $\langle f, \mathscr{E}\rangle$ is the edge $\langle f \wedge g, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$ because $f$ is ranked, according to (15), and in case also $d(\langle f, \mathscr{E}\rangle) \notin \{\blacktriangle, \blacktriangledown\}$ the same edge is generated (according to rule (11)).

□

Using this lemma we show that the solution of a formula evaluated in the context of a BES and the formula obtained from its node in a structure graph coincide.

**Lemma 3.12.** *Let $\mathscr{E}$ be a BES, $\eta$ an environment, such that $\eta(Y) = \eta(X_{\langle Y, \mathscr{E}\rangle})$ for all $Y \in \mathsf{bnd}(\mathscr{E})$. Let $f$ be a formula, such that $\mathsf{occ}(f) \subseteq \{Y \mid X_{\langle Y, \mathscr{E}\rangle} \in \mathsf{bnd}(\beta(\langle f, \mathscr{E}\rangle)) \cup \mathsf{free}(\beta(\langle f, \mathscr{E}\rangle))\}$. Then it holds that $[\![f]\!]\eta = [\![\varphi(\langle f, \mathscr{E}\rangle)]\!]\eta$.*

*Proof.* Let $\mathscr{E}$ be this BES, and $f$ a formula, and let $\eta(Y) = \eta(\langle X, \mathscr{E}\rangle)$. Assume that $\mathsf{occ}(f) \subseteq \{Y \mid X_{\langle Y, \mathscr{E}\rangle} \in \mathsf{bnd}(\beta(\langle f, \mathscr{E}\rangle)) \cup \mathsf{free}(\beta(\langle f, \mathscr{E}\rangle))\}$. We show by induction on the structure of $f$ that $[\![f]\!]\eta = [\![\varphi(\langle f, \mathscr{E}\rangle)]\!]\eta$.

- $f = \mathsf{true}$. By definition of $\varphi$, $[\![\varphi(\langle \mathsf{true}, \mathscr{E}\rangle)]\!]\eta = [\![\mathsf{true}]\!]\eta$.

- $f = \mathsf{false}$. Analogous to the previous case.

- $f = Y$. We distinguish two cases, either $Y$ is bound, or $Y$ is free:

– $Y$ is bound, *i.e.*, $X_{\langle Y,\mathscr{E}\rangle} \in \mathsf{bnd}(\beta(\langle f,\mathscr{E}\rangle))$. We derive:

$$\begin{array}{ll} & [\![\varphi(\langle Y,\mathscr{E}\rangle)]\!]\eta \\ = & \{X_{\langle Y,\mathscr{E}\rangle} \in \beta(\langle f,\mathscr{E}\rangle), \text{ hence } \langle Y,\mathscr{E}\rangle \in \mathsf{dom}(r), \text{ use definition of } \varphi\} \\ & [\![X_{\langle Y,\mathscr{E}\rangle}]\!]\eta \\ = & \{\text{Semantics of BES}\} \\ & \eta(X_{\langle Y,\mathscr{E}\rangle}) \\ = & \{\text{Assumption } \eta(X_{\langle Y,\mathscr{E}\rangle}) = \eta(Y)\} \\ & \eta(Y) \\ = & \{\text{Semantics of BES}\} \\ & [\![Y]\!]\eta \end{array}$$

– $Y \in \mathsf{free}(\beta(\langle f,\mathscr{E}\rangle))$. This case is easy, as $Y \in \mathsf{free}(\beta(\langle f,\mathscr{E}\rangle))$, also $\nearrow_{\langle Y,\mathscr{E}\rangle} Y$, hence using the definition of $\varphi$ we immediately find $[\![\varphi(\langle Y,\mathscr{E}\rangle)]\!]\eta = [\![Y]\!]\eta$.

- $f = g \wedge g'$. Based on the SOS we know that $d(\langle g \wedge g',\mathscr{E}\rangle) = \blacktriangle$. As induction hypothesis we assume that the lemma holds for all subformulae. We derive:

$$\begin{array}{ll} & [\![\varphi(\langle g \wedge g',\mathscr{E}\rangle)]\!]\eta \\ = & \{\text{Lemma 3.11}\} \\ & [\![\varphi(\langle g,\mathscr{E}\rangle) \wedge \varphi(\langle g',\mathscr{E}\rangle)]\!]\eta \\ = & \{\text{Semantics of BES}\} \\ & [\![\varphi(\langle g,\mathscr{E}\rangle)]\!]\eta \wedge [\![\varphi(\langle g',\mathscr{E}\rangle)]\!]\eta \;\; ll \\ = & \{\text{Induction hypothesis}\} \\ & [\![g]\!]\eta \wedge [\![g']\!]\eta \\ = & \{\text{Semantics of BES}\} \\ & [\![g \wedge g']\!]\eta \end{array}$$

- $f = g \vee g'$. Analogous to the previous case.

□

All of the above ultimately allows us to prove that the right hand side of an equation in a BES, and the corresponding formula obtained from its structure graph, coincide.

**Proposition 3.13.** *Let $\mathscr{E}$ be a BES such that $\sigma Y = f \in \mathscr{E}$. Then for all environments $\eta$ for which $\eta(Y) = \eta(X_{\langle Y,\mathscr{E}\rangle})$, $[\![f]\!]\eta = [\![\mathsf{rhs}(\langle Y,\mathscr{E}\rangle)]\!]\eta$.*

*Proof.* The proof proceeds by a distinction on the cases of $\mathsf{rhs}(\langle Y,\mathscr{E}\rangle)$. We show the case for $d(\langle Y,\mathscr{E}\rangle) = \blacktriangle$, the others are completely analogous.

If $d(\langle Y,\mathscr{E}\rangle) = \blacktriangle$, then $[\![\mathsf{rhs}(\langle Y,\mathscr{E}\rangle)]\!]\eta = [\![\varphi(\langle f,\mathscr{E}\rangle)]\!]\eta$ according to Lemma 3.9, using that $\sigma Y = f \in \mathscr{E}$. Using Lemma 3.12 we find that this is equivalent to $[\![f]\!]\eta$. □

Finally we show that evaluating a formula $f$ in a BES $\mathscr{E}$, and evaluating the formula $\varphi(\langle f,\mathscr{E}\rangle)$ in the BES $\beta(\langle f,\mathscr{E}\rangle)$ are equivalent.

**Theorem 3.14.** *Let $\mathscr{E}$ be a BES and $\eta$ an environment. Then for all formulae $f$ it holds that $[\![f]\!][\![\mathscr{E}]\!]\eta = [\![\varphi(\langle f,\mathscr{E}\rangle)]\!][\![\beta(\langle f,\mathscr{E}\rangle)]\!]\eta$.*

*Proof.* Let $\mathscr{F}$ abbreviate the equation system $\beta(\langle f, \mathscr{E}\rangle)$, and let $g$ abbreviate the formula $\varphi(\langle f, \mathscr{E}\rangle)$. By construction, $\mathscr{F}$ consists of equations of the form $\sigma X_{\langle Z,\mathscr{E}\rangle} = \mathsf{rhs}(\langle Z, \mathscr{E}\rangle)$, for $Z \in \mathsf{bnd}(\mathscr{E})$.

Denote the free-variable closure of $\mathscr{E}$, $\mathscr{F}$, $f$ and $g$, using $\eta$ by $\mathscr{E}_c$, $\mathscr{F}_c$, $f_c$ and $g_c$, respectively. According to Corollary 2.30, we have $[\![\mathscr{E}_c]\!] = [\![\mathscr{E}]\!]\eta$, and $[\![\mathscr{F}_c]\!] = [\![\mathscr{F}]\!]\eta$, and, likewise, $[\![f]\!][\![\mathscr{E}]\!]\eta = [\![f_c]\!][\![\mathscr{E}_c]\!]$ and $[\![g]\!][\![\mathscr{F}]\!]\eta = [\![g_c]\!][\![\mathscr{F}_c]\!]$. Let $\mathscr{E}'$ be the equation system obtained by merging all equations of $\mathscr{E}_c$ and $\mathscr{F}_c$, such that:

1. $\mathsf{rank}_{\mathscr{E}'}(X) = \mathsf{rank}_{\mathscr{E}}(X)$ for all $X \in \mathsf{bnd}(\mathscr{E}_c)$;

2. $\mathsf{rank}_{\mathscr{E}'}(X_{\langle Z,\mathscr{E}\rangle}) = \mathsf{rank}_{\mathscr{E}}(Z)$ for all $X_{\langle Z,\mathscr{E}\rangle} \in \mathsf{bnd}(\mathscr{F}_c)$.

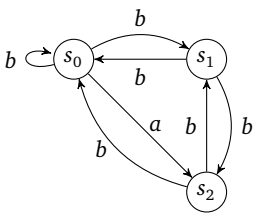
Observe that the resulting $\mathscr{E}'$ is well-formed, since we have $\mathsf{bnd}(\mathscr{E}_c) \cap \mathsf{bnd}(\mathscr{F}_c) = \emptyset$. Moreover, since $\mathsf{bnd}(\mathscr{E}_c) \cap \mathsf{occ}(\mathscr{F}_c) = \mathsf{bnd}(\mathscr{F}_c) \cap \mathsf{occ}(\mathscr{E}_c) = \emptyset$ also $[\![\mathscr{E}']\!] = [\![\mathscr{E}_c]\!][\![\mathscr{F}_c]\!]$ according to Lemma 2.28, *i.e.*, we can find the solution to $\mathscr{E}_c$ and $\mathscr{F}_c$ by solving $\mathscr{E}'$. Using Proposition 3.13, we find that the relation $R$, defined as

$$R = \{(Z, X_{\langle Z,\mathscr{E}\rangle}), (X_{\langle Z,\mathscr{E}\rangle}, Z) \mid X_{\langle Z,\mathscr{E}\rangle} \in \mathscr{F}_c\}$$

is a consistent correlation. According to Theorem 2.33 we therefore find that for all $X_{\langle Z,\mathscr{E}\rangle} \in \mathsf{bnd}(\mathscr{F}_c)$, we have $[\![Z]\!][\![\mathscr{E}']\!] = [\![X_{\langle Z,\mathscr{E}\rangle}]\!][\![\mathscr{E}']\!]$. More specifically, $[\![Z]\!][\![\mathscr{E}_c]\!] = [\![X_{\langle Z,\mathscr{E}\rangle}]\!][\![\mathscr{F}_c]\!]$, and hence also $[\![f_c]\!][\![\mathscr{E}_c]\!] = [\![g_c]\!][\![\mathscr{F}_c]\!]$. Our claim now follows. □

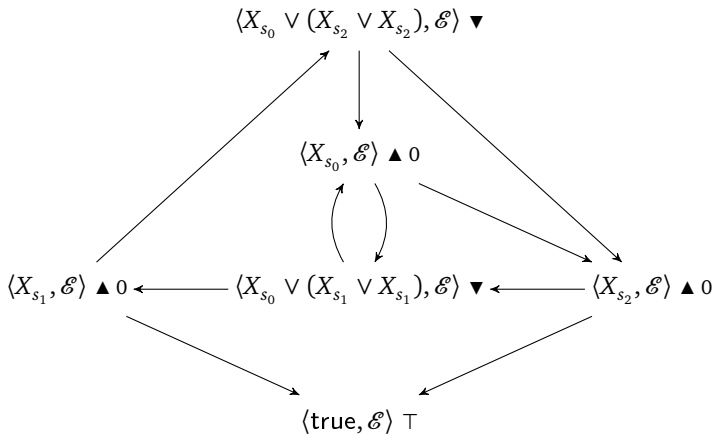
We illustrate the various translations described in this section through the following example.

**Example 3.15.** Consider the labelled transition system $L$ given below.

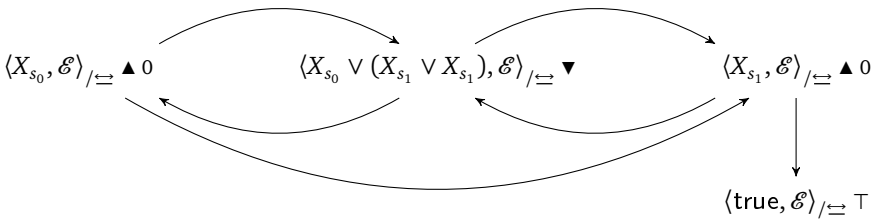



Let $\phi = \nu X.[a]X \wedge \langle b\rangle X$ be the formula that encodes that along every $a$ path an infinite $b$ path is enabled. Consider the following equation system $\mathscr{E} = \mathsf{E}^L(\phi)$, encoding the model checking problem whether $L$ satisfies $\phi$, together with the structure graph $\langle X_{s_0}, \mathscr{E}\rangle$:

$$\begin{array}{lcl} (\nu X_{s_0} & = & (X_{s_2} \wedge X_{s_2}) \wedge (X_{s_0} \vee (X_{s_1} \vee X_{s_1}))) \\ (\nu X_{s_1} & = & \mathsf{true} \wedge (X_{s_0} \vee (X_{s_2} \vee X_{s_2}))) \\ (\nu X_{s_2} & = & \mathsf{true} \wedge (X_{s_0} \vee (X_{s_1} \vee X_{s_1}))) \end{array}$$




Observe that the above structure graph can be minimised with respect to bisimilarity, identifying vertices $\langle X_{s_1}, \mathscr{E}\rangle$ and $\langle X_{s_2}, \mathscr{E}\rangle$, as well as $\langle X_{s_0} \vee (X_{s_1} \vee X_{s_1}), \mathscr{E}\rangle$ and $\langle X_{s_0} \vee (X_{s_2} \vee X_{s_2}), \mathscr{E}\rangle$. This leads to the following bisimilar, minimal structure graph:




The above structure graph induces the following equation system, using the translation of Definition 3.8.

$$(\nu X_{\langle X_{s_0}, \mathscr{E}\rangle_{/\leftrightarrows}} = (X_{\langle X_{s_0}, \mathscr{E}\rangle_{/\leftrightarrows}} \vee (X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}} \vee X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}})) \wedge (X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}} \wedge X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}}))$$
$$(\nu X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}} = (X_{\langle X_{s_0}, \mathscr{E}\rangle_{/\leftrightarrows}} \vee (X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}} \vee X_{\langle X_{s_1}, \mathscr{E}\rangle_{/\leftrightarrows}})) \wedge (\mathsf{true} \wedge \mathsf{true}))$$

The size of the original structure graph is 6. By comparison, the size of the minimal structure graph is 4. As will become clear in Section 3.3.2, solving the above equation system enables one to deduce the solution to the original equation system.

### 3.1.3 Solution Equivalence of Structure Graphs

The interpretation of a structure graph as a BES using functions $\varphi$ and $\beta$ gives rise to the notion of *solution equivalence*. We say that two BESsy structure graphs are solution equivalent if the formulae obtained from the initial vertices in these structure graphs, evaluated in the corresponding BESs, have the same solution. Formally this is defined as follows.

**Definition 3.16.** Let $t$ and $t'$ be arbitrary BESsy structure graphs. Structure graphs $t$ and $t'$ are solution equivalent, denoted $t \equiv t'$ if and only if, for all environments $\eta$:

$$[\![\varphi(t)]\!]\,[\![\beta(t)]\!]\,\eta = [\![\varphi(t')]\!]\,[\![\beta(t')]\!]\,\eta$$

In a possible lattice of equivalence relations on BESsy structure graphs, $\equiv$ is the coarsest equivalence relation of interest. Deciding $\equiv$ is in NP $\cap$ co-NP.

## 3.2 Normalisation of Structure Graphs

In BESsy structure graphs, a vertex that is decorated by a rank typically represents a proposition variable that occurs at the left-hand side of some equation in the associated equation system, whereas the non-ranked vertices can occur as subterms in right-hand sides of equations with mixed occurrences of $\wedge$ and $\vee$. *Normalisation* of a structure graph assigns a rank to each non-ranked vertex that has successors. The net effect of this operation is that the structure graph obtained this way induces an equation system in simple form. In choosing the rank, one has some degree of freedom; an effective and sound strategy is to ensure that all equations in the associated equation system end up in the very last block. This is typically achieved by assigning 0 as a rank. This normalisation is achieved using the following set of deduction rules.

$$(25)\ \frac{t\blacktriangle}{\mathsf{norm}(t)\blacktriangle} \qquad (26)\ \frac{t\blacktriangledown}{\mathsf{norm}(t)\blacktriangledown} \qquad (27)\ \frac{t \to t'}{\mathsf{norm}(t) \to \mathsf{norm}(t')}$$

$$(28)\ \frac{t\top}{\mathsf{norm}(t)\top} \qquad (29)\ \frac{t\bot}{\mathsf{norm}(t)\bot} \qquad (30)\ \frac{t \nearrow_X}{\mathsf{norm}(t) \nearrow_X}$$

$$(31)\ \frac{t \Cap n}{\mathsf{norm}(t) \Cap n} \qquad (32)\ \frac{t \not\blacktriangle \quad t \to t'}{\mathsf{norm}(t) \Cap 0}$$

The last deduction rule expresses that in case a vertex $t$ does not have a rank, rank 0 is associated to the normalised version of $t$, provided, of course, that the vertex has a successor. Observe that normalisation preserves BESsyness of the structure graph, *i.e.*, any BESsy structure graph that is normalised again yields a BESsy structure graph.

**Proposition 3.17.** *Let $t$ be an arbitrary BESsy structure graph.*

1. $\varphi(\mathsf{norm}(t)) \in \mathcal{X} \cup \{\mathsf{true}, \mathsf{false}\}$*;*
2. $\beta(\mathsf{norm}(t))$ *is in simple form.*

*Proof.* Follows immediately from the definitions. □

The well-definedness of the extended SOS is obtained by adapting the stratification from the previous SOS by requiring that $t \Cap n$ is larger than $u \Cap m$ in all cases where the number of occurrences of norm in $t$ is larger than in $u$.

The lemmata below formalise that the solution to an equation system that is induced by a BESsy structure graph, is preserved and reflected by the equation system associated to the normalised counterpart of that structure graph.

**Lemma 3.18.** *Let $t$ be a BESsy structure graph. Then, there is a total injective mapping $h : \mathsf{bnd}(\beta(t)) \to \mathsf{bnd}(\beta(\mathsf{norm}(t)))$, such that for all $\eta$:*

$$\forall X \in \mathsf{bnd}(\beta(t)) : [\![\beta(t)]\!]\eta(X) = [\![\beta(\mathsf{norm}(t))]\!]\eta(h(X))$$

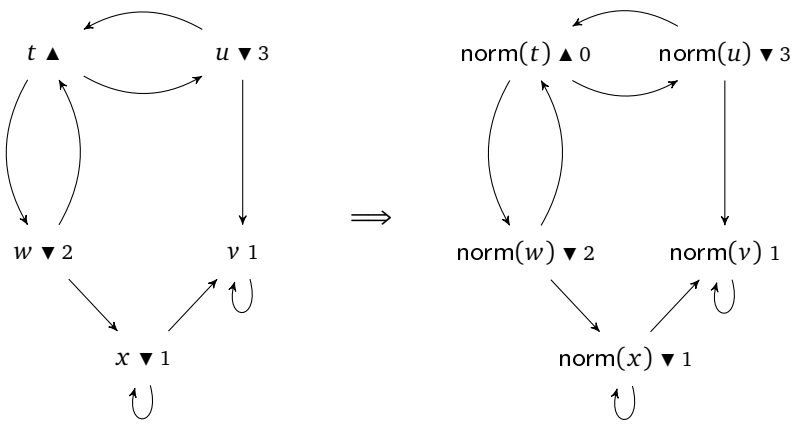
*Proof.* Observe that for each ranked vertex $u$ in $t$, vertex $\mathsf{norm}(u)$ has the same rank in $\mathsf{norm}(t)$. Following Definition 3.8, these vertices both induce equations in the equation systems that appear in the same block of identical fixed point equations. All unranked vertices $u'$ in $t$ that are ranked in $\mathsf{norm}(t)$, induce $\nu$-equations at the end of the equation system induced by $\mathsf{norm}(t)$. References to these latter equations can be eliminated, following Lemmata 2.25 and 2.26. □

**Lemma 3.19.** *Let $t$ be a BESsy structure graph. Then $t \equiv \mathsf{norm}(t)$.*

*Proof.* Follows from Lemma 3.18. □

The example below illustrates an application of normalisation, and it provides a demonstration of the above lemmata and its implications.

**Example 3.20.** The BESsy structure graph depicted at the left contains a single vertex that is not decorated with a rank. Normalisation of this structure graph yields the structure graph depicted at the right.




Assuming that vertex $t$ is the initial vertex, $\beta(t)$ is as follows:

$$
\begin{aligned}
&(\mu X_u = (X_u \wedge (X_w \wedge X_w)) \vee (X_v \vee X_v))\\
&(\nu X_w = (X_u \wedge (X_w \wedge X_w)) \vee (X_x \vee X_x))\\
&(\mu X_v = X_v)\\
&(\mu X_x = X_v \vee (X_x \vee X_x))
\end{aligned}
$$

$\beta(\mathsf{norm}(t))$ has similar top-level logical operands as $\beta(t)$, but contains an extra greatest fixed point equation trailing the other four equations, and references to this equation:

$$
\begin{aligned}
&(\mu X_{\mathsf{norm}(u)} = X_{\mathsf{norm}(t)} \vee (X_{\mathsf{norm}(v)} \vee X_{\mathsf{norm}(v)}))\\
&(\nu X_{\mathsf{norm}(w)} = X_{\mathsf{norm}(t)} \vee (X_{\mathsf{norm}(x)} \vee X_{\mathsf{norm}(x)}))\\
&(\mu X_{\mathsf{norm}(v)} = X_{\mathsf{norm}(v)})\\
&(\mu X_{\mathsf{norm}(x)} = X_{\mathsf{norm}(v)} \vee (X_{\mathsf{norm}(x)} \vee X_{\mathsf{norm}(x)}))\\
&(\nu X_{\mathsf{norm}(t)} = X_{\mathsf{norm}(u)} \wedge (X_{\mathsf{norm}(w)} \wedge X_{\mathsf{norm}(w)}))
\end{aligned}
$$

According to Lemma 3.18, there is an injection $h : \mathsf{bnd}(\beta(t)) \to \mathsf{bnd}(\beta(\mathsf{norm}(t)))$, such that for all $X \in \mathsf{bnd}(\beta(t))$, we have $[\![\beta(t)]\!](X) = [\![\beta(\mathsf{norm}(t))]\!](h(X))$; $h(X_z) = X_{\mathsf{norm}(z)}$

for $z \in \{u, v, w, x\}$ is such an injection. Following Lemma 3.19, we find $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = [\![X_u \wedge (X_w \wedge X_w)]\!]\,[\![\beta(t)]\!] = [\![X_{\mathsf{norm}(t)}]\!]\,[\![\beta(\mathsf{norm}(t))]\!] = [\![\varphi(\mathsf{norm}(t))]\!]\,[\![\beta(\mathsf{norm}(t))]\!]$, which is false.

## 3.3 Properties of Strong Bisimilarity of Structure Graphs

Now that we have introduced structure graphs, the concept of bisimilarity for structure graphs, and normalisation, it is time to study the effects of bisimilarity on the normalisation of structure graphs. In this section we first show that bisimulation preserves BESsyness. We then continue with a number of congruence results. Finally we determine whether bisimilarity of structure graphs indeed reflects associativity as we desired in Section 3.1.1.

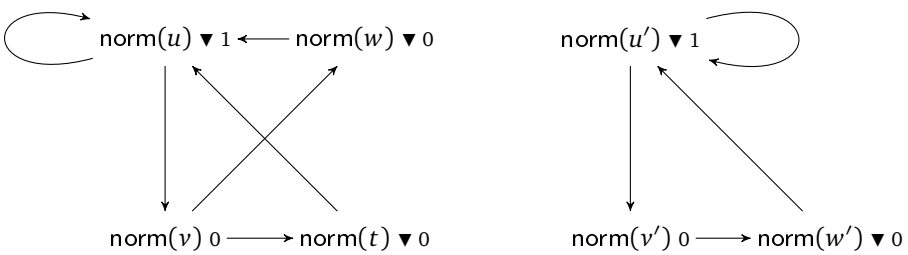
BESsyness is preserved under bisimilarity, as shown in the following lemma.

**Lemma 3.21.** *Let $\mathcal{G}$ be a BESsy structure graph, then $\mathcal{G}_{/\leftrightarrows}$ is BESsy.*

*Proof.* This follows immediately from the transfer conditions of bisimilarity. □

The converse, however, does not hold, as is witnessed by the following example.

**Example 3.22.** Consider the normalised structure graph to the left. Observe that this graph is not BESsy because node $\mathsf{norm}(v)$ has multiple successors, but is not decorated by ▲ or ▼. Its bisimulation quotient, shown to the right, is BESsy.




### 3.3.1 Congruence

Bisimilarity on structure graphs is a congruence for $\wedge$ and $\vee$, as shown in Lemma 3.23. This allows us to reason about bisimilarity of vertices representing conjuncts and disjuncts in terms of bisimilarity of vertices representing their subformulae.

**Lemma 3.23.** *Let $\mathcal{E}$ be an equation system. Let $f$, $f'$, $g$ and $g'$ be arbitrary proposition formulae such that $\langle f, \mathcal{E}\rangle \leftrightarrows \langle f', \mathcal{E}\rangle$ and $\langle g, \mathcal{E}\rangle \leftrightarrows \langle g', \mathcal{E}\rangle$. Then the following hold:*

$$\langle f \wedge g, \mathcal{E}\rangle \leftrightarrows \langle f' \wedge g', \mathcal{E}\rangle, \quad \langle f \vee g, \mathcal{E}\rangle \leftrightarrows \langle f' \vee g', \mathcal{E}\rangle$$

*Proof.* Suppose that bisimilarity of $\langle f, \mathcal{E}\rangle$ and $\langle f', \mathcal{E}\rangle$ is witnessed by $R$ and the bisimilarity of $\langle g, \mathcal{E}\rangle$ and $\langle g', \mathcal{E}\rangle$ is witnessed by $S$. The relation $\{(\langle f \wedge g, \mathcal{E}\rangle, \langle f' \wedge g', \mathcal{E}\rangle)\} \cup R \cup S$ is a bisimulation relation that proves bisimilarity of $\langle f \wedge g, \mathcal{E}\rangle$ and $\langle f' \wedge g', \mathcal{E}\rangle$. Similarly, $\{(\langle f \vee g, \mathcal{E}\rangle, \langle f' \vee g', \mathcal{E}\rangle)\} \cup R \cup S$ is a bisimulation relation that proves bisimilarity of $\langle f \vee g, \mathcal{E}\rangle$ and $\langle f' \vee g', \mathcal{E}\rangle$. □

The following lemma indicates that bisimilarity on structure graphs indeed respects logical equivalences such as commutativity, associativity and a weak form of idempotence for the $\wedge$ and $\vee$ operators.

**Lemma 3.24.** *Let $\mathscr{E}$ be an equation system. Let $f$, $f'$, and $f''$ be arbitrary proposition formulae. Then the following hold:*

$$\begin{array}{rcl}
\langle (f \wedge f') \wedge f'', \mathscr{E}\rangle & \leftrightarrows & \langle f \wedge (f' \wedge f''), \mathscr{E}\rangle, \\
\langle (f \vee f') \vee f'', \mathscr{E}\rangle & \leftrightarrows & \langle f \vee (f' \vee f''), \mathscr{E}\rangle, \\
\langle f \wedge f', \mathscr{E}\rangle & \leftrightarrows & \langle f' \wedge f, \mathscr{E}\rangle, \\
\langle f \vee f', \mathscr{E}\rangle & \leftrightarrows & \langle f' \vee f, \mathscr{E}\rangle, \\
\langle (f \wedge f) \wedge f', \mathscr{E}\rangle & \leftrightarrows & \langle f \wedge f', \mathscr{E}\rangle, \\
\langle (f \vee f) \vee f', \mathscr{E}\rangle & \leftrightarrows & \langle f \vee f', \mathscr{E}\rangle
\end{array}$$

*Proof.* The proofs are easy. For example, the bisimulation relation that witnesses bisimilarity of $\langle (f \wedge f') \wedge f'', \mathscr{E}\rangle$ and $\langle f \wedge (f' \wedge f''), \mathscr{E}\rangle$ is the relation that relates all formulae of the form $\langle (g \wedge g') \wedge g'', \mathscr{E}\rangle$ and $\langle g \wedge (g' \wedge g''), \mathscr{E}\rangle$ and additionally contains the identity relation on structure graphs. Proofs of the 'transfer conditions' are easy as well. As an example, suppose that $\langle (g \wedge g') \wedge g'', \mathscr{E}\rangle \to \langle h, \mathscr{E}\rangle$ for some formula $h$. In case this transition is due to $\langle g \wedge g', \mathscr{E}\rangle$▲ and $\langle g \wedge g', \mathscr{E}\rangle \to \langle h, \mathscr{E}\rangle$, one of the cases that occurs for $\langle g \wedge g', \mathscr{E}\rangle \to \langle h, \mathscr{E}\rangle$ is that $\langle g, \mathscr{E}\rangle$▲ and $\langle g, \mathscr{E}\rangle \to \langle h, \mathscr{E}\rangle$. We obtain $\langle g \wedge (g' \wedge g''), \mathscr{E}\rangle \to \langle h, \mathscr{E}\rangle$. Since $\langle h, \mathscr{E}\rangle$ and $\langle h, \mathscr{E}\rangle$ are related, this finishes the proof of the transfer condition in this case. All other cases are similar or at least equally easy. □

**Corollary 3.25.** *Let $\mathscr{E}$ be an equation system. Let $F$ and $G$ be arbitrary finite sets of proposition formulae such that (1) for all $f \in F$ there exists $g \in G$ with $\langle f, \mathscr{E}\rangle \leftrightarrows \langle g, \mathscr{E}\rangle$, and, vice versa, (2) for all $g \in G$ there exists $f \in F$ with $\langle g, \mathscr{E}\rangle \leftrightarrows \langle f, \mathscr{E}\rangle$. Then, $\langle \bigsqcap F, \mathscr{E}\rangle \leftrightarrows \langle \bigsqcap G, \mathscr{E}\rangle$ and $\langle \bigsqcup F, \mathscr{E}\rangle \leftrightarrows \langle \bigsqcup G, \mathscr{E}\rangle$.*

*Proof.* The corollary follows immediately from the congruence of $\wedge$ and $\vee$ (Lemma 3.23) and commutativity and associativity of those (Lemma 3.24). □

Idempotence of $\wedge$ and $\vee$, and more involved logical equivalences such as distribution and absorption are not captured by isomorphism or even bisimilarity on the structure graphs. The reason is that, for an arbitrary equation system $\mathscr{E}$ and variable $X$, the vertex associated with $\langle X \wedge X, \mathscr{E}\rangle$ will be decorated by ▲, in contrast to the vertex associated with $\langle X, \mathscr{E}\rangle$.

The below proposition states that bisimilarity on structure graphs is a congruence for normalisation.

**Proposition 3.26.** *Let $t, t'$ be arbitrary, but bisimilar structure graphs. Then* $\mathsf{norm}(t) \leftrightarrows \mathsf{norm}(t')$.

*Proof.* Let $R$ be a bisimulation relation witnessing $t \leftrightarrows t'$. We define the relation $R_n$ as $\{(\mathsf{norm}(u), \mathsf{norm}(u')) \mid (u, u') \in R\}$. Then $R_n$ is a bisimulation relation witnessing $\mathsf{norm}(t) \leftrightarrows \mathsf{norm}(t')$. □

Observe that normalising the same structure graph multiple times still leads to bisimilar structure graphs, *i.e.*, $\mathsf{norm}(\mathsf{norm}(t)) \leftrightarrows \mathsf{norm}(t)$.

Ultimately, the above proposition implies that the simple form is not harmful from a strong bisimulation perspective: normalisation does not lead to larger quotients of structure graphs. This addresses the question concerning the effect of normalisation on the reductive capabilities of bisimulation. Formally, we have:

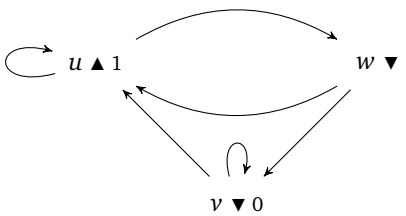
**Theorem 3.27.** *Let $t$ be an arbitrary structure graph. Then $t_{/\leftrightarrows}$ is at least as large as $\mathsf{norm}(t)_{/\leftrightarrows}$.*

*Proof.* The theorem follows immediately from the fact that $\mathsf{norm}(t)$ and $t$ are equal in size, and Proposition 3.26. □

The example below illustrates that normalisation can in fact sometimes be beneficial for the minimising capabilities of bisimulation.

**Example 3.28.** Consider the following equation system $\mathscr{E}$, with the associated structure graph $\langle X, \mathscr{E}\rangle$ (represented by vertex $u$) depicted next to it.

$(\mu X = X \wedge (Y \vee X))$
$(\nu Y = Y \vee X)$




Clearly, the structure graph is already minimal. Normalisation *upcasts* vertex $w$ to a ranked vertex, assigning rank 0 to it. It is then easy to check that vertices $\mathsf{norm}(w)$ and $\mathsf{norm}(v)$ are bisimilar. Hence, the quotient of $\mathsf{norm}(u)$ has size 2, compared to size 3 for $u$.

### 3.3.2 Bisimilarity Implies Solution Equivalence

In this section we state one of our main results, proving that equation systems corresponding to bisimilar BESsy structure graphs essentially have the same solution. This allows one to safely use bisimulation minimisation of the structure graph, and solve the equation system induced by the minimal structure graph instead. Before we give our main theorem, we first lift the results that allow constructing purely conjunctive and purely disjunctive equation systems, see Proposition 2.23, from equation systems to structure graphs.

**Definition 3.29.** Let $\langle T, t, \rightarrow, d, r, \nearrow\rangle$ be a structure graph. A partial function $\gamma : T \mapsto T$ is a $\bullet$-*choice function*, with $\bullet \in \{\blacktriangle, \blacktriangledown\}$, when both:

- $\mathsf{dom}(\gamma) = \{u \in T \mid d(u) = \bullet \wedge u \rightarrow\}$;
- $u \rightarrow \gamma(u)$ for all $u \in \mathsf{dom}(\gamma)$.

Given a $\bullet$-choice function $\gamma$, with $\bullet \in \{\blacktriangle, \blacktriangledown\}$, for a structure graph, we can obtain a new structure graph by choosing one successor among the successors for vertices decorated with a $\bullet$, *viz.*, the one prescribed by $\gamma$. This is formalised next.

**Definition 3.30.** Let $\mathcal{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be an arbitrary structure graph. Let $\bullet \in \{\blacktriangle, \blacktriangledown\}$, and $\gamma$ a $\bullet$-choice function. The structure graph $\mathcal{G}_\gamma$, obtained by applying the $\bullet$-choice function $\gamma$ on $\mathcal{G}$, is defined as the six-tuple $\langle T, t, \rightarrow_\gamma, d_\gamma, r, \nearrow \rangle$, where:

- for all $u \notin \mathsf{dom}(\gamma)$, $u \rightarrow_\gamma u'$ if and only if $u \rightarrow u'$;
- for all $u \in \mathsf{dom}(\gamma)$, only $u \rightarrow_\gamma \gamma(u)$;
- $d_\gamma(t) = d(t)$ and $\mathsf{dom}(d_\gamma) = \{u \mid d(u) \neq \bullet\}$

Observe that a structure graph obtained by applying a $\blacktriangle$-choice function entails a structure graph in which no vertex is labelled with $\blacktriangle$. Similarly, applying a $\blacktriangledown$-choice function yields a structure graph without $\blacktriangledown$ labelled vertices.

*Property* 3.31. Let $t$ be an arbitrary BESsy structure graph. Assume an arbitrary $\bullet$-choice function $\gamma$ on $t$. Then $\mathsf{norm}(t_\gamma)$ is again BESsy.

The effect that applying, *e.g.*, a $\blacktriangle$-choice function has on the solution of the equation system associated to the structure graph to which it is applied, is characterised by the proposition below, which is the modification of Proposition 2.23 to structure graphs.

**Proposition 3.32.** *Let $t$ be a normalised, BESsy structure graph, with no vertex labelled $\nearrow$.*

1. *For all $\blacktriangle$-choice functions $\gamma$ applied to $t$, we have $[\![\beta(t)]\!] \sqsubseteq [\![\beta(t_\gamma)]\!]$;*
2. *There exists a $\blacktriangle$-choice function $\gamma$, such that $[\![\beta(t)]\!] = [\![\beta(t_\gamma)]\!]$.*
3. *For all $\blacktriangledown$-choice functions $\gamma$ applied to $t$, we have $[\![\beta(t)]\!] \sqsupseteq [\![\beta(t_\gamma)]\!]$;*
4. *There exists a $\blacktriangledown$-choice function $\gamma$, such that $[\![\beta(t)]\!] = [\![\beta(t_\gamma)]\!]$.*

*Proof.* Follows immediately from Proposition 2.23, and the correspondence between structure graphs and Boolean Equation Systems. □

In some cases, *viz.*, when a structure graph is void of any vertices labelled $\blacktriangledown$ or void of vertices labelled $\blacktriangle$, the solution of an equation system associated to a structure graph can be characterised by the structure of the graph. While one could consider these to be degenerate cases, they are essential in our proof of the main theorem in this section. A key concept used in characterising the solution of equation systems in these degenerate cases is that of a $\nu$-dominated lasso, and its dual, $\mu$-dominated lasso.

**Definition 3.33.** Let $t$ be a BESsy structure graph. A lasso starting in $t$ is a finite sequence $t_0, t_1, \ldots, t_n$, satisfying $t_0 = t$, $t_n = t_j$ for some $j \leq n$, and for each $1 \leq i \leq n$, $t_{i-1} \rightarrow t_i$. A lasso is said to be $\nu$-dominated if $\max\{r(t_i) \mid j \leq i \leq n\}$ is even; otherwise it is $\mu$-dominated.

The following lemma is loosely based on [Kei06, Lemmata 40 and 41].

**Lemma 3.34.** *Let $t$ be a normalised, BESsy structure graph in which no vertex is labelled with $\nearrow$. Then:*

1. *if no vertex in $t$ is labelled with $\blacktriangle$ then $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{true}$ if and only if some lasso starting in $t$ is $\nu$-dominated, or some maximal, finite path starting in $t$ terminates in a vertex labelled with $\top$;*

2. *if no vertex in $t$ is labelled with $\blacktriangledown$ then $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{false}$ if and only if some lasso starting in $t$ is $\mu$-dominated, or some maximal, finite path starting in $t$ terminates in a vertex labelled with $\bot$*

*Proof.* We only consider the first statement; the proof of the second statement is dual. Observe that since no vertex in $t$ is labelled with $\blacktriangle$, $\varphi(u) \neq \bigsqcap\{u_1, \ldots, u_n\}$ for all $u$. We distinguish two cases:

1. Assume there is a $\nu$-dominated lasso $t_0, t_1, \ldots, t_n$, starting in $t$. BESsyness of $t$ implies that there is a ranked vertex $t_i$ on the cycle of the lasso. Without loss of generality assume that $t_i$ has the highest rank on the cycle of the $\nu$-dominated lasso. By definition, this highest rank is even. This means that it induces an equation $\nu X_{t_i} = g_i$ in $\beta(t)$, that precedes all other equations $\sigma X_{t_k} = g_k$ induced by the other vertices on the cycle. Consider the path snippet starting in $t_i$, leading to $t_i$ again: $t_i, t_{i+1}, \ldots, t_{n-1}, t_j, t_{j+1}, t_{i-1}$. Lemma 2.26, *i.e.*, Gauß elimination, allows one to substitute $g_{i+1}$ for $X_{t_{i+1}}$ in the equation for $X_{t_i}$, yielding $\nu X_{t_i} = g_i[X_{t_{i+1}} := g_{i+1}]$. Repeatedly applying Gauß elimination on the path snippet ultimately allows one to rewrite $\nu X_{t_i} = g_i$ to $\nu X_{t_i} = g_i' \vee X_{t_i}$, since $X_{t_{i-1}}$ depends on $X_{t_i}$ again, and none of the formulae is conjunctive. The solution to $\nu X_{t_i} = g_i' \vee X_{t_i}$ is easily seen to be $X_{t_i} = \mathsf{true}$. This solution ultimately propagates through the entire lasso, and back to $t$, leading to $\varphi(t) = X_t = \mathsf{true}$.

2. Suppose there is a finite path $t_0, t_1, \ldots, t_n$ starting in $t$, where $t_n$ is labelled with $\top$. This means that there is an equation $\sigma X_{t_n} = \mathsf{true}$ on which $X_t$ depends. As the equation $\sigma X_{t_n} = \mathsf{true}$ is solved, we may immediately substitute the solution in all other formulae on the path. As none of the formulae is conjunctive, we find $\varphi(t) = \mathsf{true}$.

Conversely, observe that due to Proposition 3.32, there is a structure graph $t_{\blacktriangledown}$, void of any vertices labelled $\blacktriangledown$, that has an equation system associated to it with a solution equivalent to that of the equation system associated to $t$. This means that $t_{\blacktriangledown}$ has no branching structure, but is necessarily a set of lassos and maximal, finite paths. In case the initial vertex of $t$ is on a lasso, $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{true}$ holds because the cycle on the lasso has an even highest rank. In the other case, $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{true}$ can only be the case because ultimately $t_{\blacktriangledown}$ leads to a vertex labelled $\mathsf{true}$. □

We prove that, for BESsy structure graphs that do not have vertices labelled with $\nearrow$, bisimilar structure graphs are also solution equivalent.

**Lemma 3.35.** *Let $t$, $t'$ be normalised BESsy structure graphs in which no vertex is labelled with $\nearrow$. Assume $t$ is minimal w.r.t strong bisimilarity. Then $t \leftrightarrows t'$ implies $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = [\![\varphi(t')]\!]\,[\![\beta(t')]\!]$.*

*Proof.* The case where the initial vertex of $t$ is decorated with a $\top$ or $\bot$ is trivial and therefore omitted. Assume that the initial vertex of $t$ is not decorated with $\top$ nor $\bot$. Suppose that $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{true}$. By Proposition 3.32 we know that there is a $\blacktriangledown$-choice function $\gamma$ such that $[\![\beta(t_\gamma)]\!] = [\![\beta(t)]\!]$. We next construct a $\blacktriangledown$-choice function $\gamma'$ for $t'$ that satisfies the following condition:

$$\forall u \in \mathsf{dom}(\gamma), u' \in \mathsf{dom}(\gamma') :\ u \leftrightarrows u' \implies \gamma(u) \leftrightarrows \gamma'(u')$$

Note that the minimality of $t$ implies that $\gamma$ satisfies $\gamma(w) \leftrightarrows \gamma(w')$ for all $w \leftrightarrows w'$ with $w, w' \in \mathsf{dom}(\gamma)$. We then have $t_\gamma \leftrightarrows t_{\gamma'}$, as the choice for successors chosen in previously bisimilar $\blacktriangledown$-labelled vertices is synchronised by the $\blacktriangledown$-choice function. Because of this bisimilarity and the finiteness of $t'$, any $\nu$-dominated lasso starting in a vertex $u$ reachable in $t$ implies the existence of a similar $\nu$-dominated lasso starting in vertices $u'$ reachable in $t'$ that are bisimilar to $u$, and, of course, also *vice versa*. Likewise for maximal finite paths. Suppose the initial vertex of $t_\gamma$ has only $\nu$-dominated lassos and finite maximal paths ending in $\top$-labelled vertices. Then, by construction, so has $t'_{\gamma'}$. This means that

$$[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = [\![\varphi(t_\gamma)]\!]\,[\![\beta(t_\gamma)]\!] =^{\dagger} \mathsf{true} = [\![\varphi(t'_{\gamma'})]\!]\,[\![\beta(t'_{\gamma'})]\!]$$

where at $^{\dagger}$, Lemma 3.34 is used. Using Proposition 3.32, we find:

$$[\![\varphi(t'_{\gamma'})]\!]\,[\![\beta(t'_{\gamma'})]\!] \Longrightarrow [\![\varphi(t')]\!]\,[\![\beta(t')]\!]$$

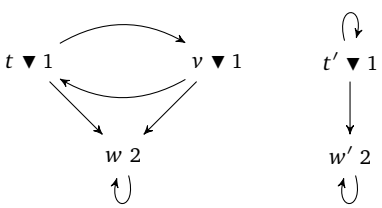
Combining the above, we can conclude that we have:

$$[\![\varphi(t')]\!]\,[\![\beta(t')]\!] = \mathsf{true}$$

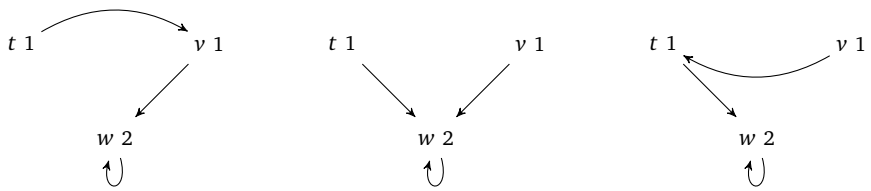
The case where $[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = \mathsf{false}$ follows the same line of reasoning, constructing a structure graph with a $\blacktriangle$-choice function $\gamma$, resulting in a structure graph containing no vertices labelled $\blacktriangle$. □

We set out to prove that arbitrary bisimilar structure graphs $t$ and $t'$ always give rise to equation systems and formulae with the same truth value. The above lemma may seem like a roundabout way in proving this property. In particular, the assumption in Lemma 3.35 that $t$ is minimal with respect to bisimilarity may seem odd. The reason for using the quotient is due to our appeal to the non-constructive Proposition 3.32, as we illustrate through the following example.

**Example 3.36.** Consider the two bisimilar BESsy structure graphs $t$ and $t'$ below:




Following Lemma 3.34, we know that all vertices will be associated to proposition variables with solution $\mathsf{true}$, as both structure graphs are normalised and contain no $\blacktriangle$-labelled vertices. Appealing to Proposition 3.32, we know that there is a structure graph $t_\blacktriangledown$ that gives rise to an equation system with the same solution as the one that can be associated to $t$. In fact, there are three choices for $t_\blacktriangledown$:



Note that all three structure graphs are associated to equation systems with the same solution as the equation system for $t$. However, while the middle structure graph would allow us to construct a $\blacktriangledown$-choice function that resolves the choice for successors for vertex $t'$, the other two structure graphs do not allow us to do so, simply because they have bisimilar vertices whose only successor leads to different equivalence classes. Such conflicts do not arise when assuming that $t$ is already minimal, in which case each vertex represents a unique class.

Regardless of the above example, we can still derive the desired result. Based on the previous lemma, the fact that bisimilarity is an equivalence relation on structure graphs and the fact that quotienting is well-behaved, we find the following theorem, which holds for arbitrary BESsy structure graphs.

**Theorem 3.37.** *Let $t, t'$ be arbitrary bisimilar BESsy structure graphs. Then for all environments $\eta$, $[\![\varphi(t)]\!]\,[\![\beta(t)]\!]\,\eta = [\![\varphi(t')]\!]\,[\![\beta(t')]\!]\,\eta$.*

*Proof.* Let $\eta$ be an arbitrary environment. Let $\overline{t}$ and $\overline{t}'$ be the structure graphs obtained from $t$ and $t'$ by replacing all decorations of the form $\nearrow_X$ of all vertices with $\top$ if $\eta(X) = \mathsf{true}$, and $\bot$ otherwise. Note that we have $\overline{t} \leftrightarrows \overline{t}'$. Based on Lemma 2.29 and Definition 3.8, we find:

$$[\![\varphi(t)]\!]\,[\![\beta(t)]\!]\,\eta = [\![\varphi(\overline{t})]\!]\,[\![\beta(\overline{t})]\!]$$

Likewise, we can derive such an equivalence for $\overline{t}'$ and $t'$. By Lemma 3.19, we find:

$$[\![\varphi(\overline{t})]\!]\,[\![\beta(\overline{t})]\!] = [\![\varphi(\mathsf{norm}(\overline{t}))]\!]\,[\![\beta(\mathsf{norm}(\overline{t}))]\!]$$

Again, a similar equivalence can be derived for $\overline{t}'$ and $\mathsf{norm}(\overline{t}')$. Observe that by Proposition 3.26, we find that $\overline{t} \leftrightarrows \overline{t}'$ implies $\mathsf{norm}(\overline{t}) \leftrightarrows \mathsf{norm}(\overline{t}')$. Observe that $\mathsf{norm}(\overline{t}) \leftrightarrows \mathsf{norm}(\overline{t})_{/\leftrightarrows} \leftrightarrows \mathsf{norm}(\overline{t}')$. Finally, since all three are still BESsy structure graphs, that furthermore do not contain vertices labelled with $\nearrow$, we can apply Lemma 3.35 twice to find:

$$\begin{aligned} & [\![\varphi(\mathsf{norm}(\overline{t}))]\!]\,[\![\beta(\mathsf{norm}(\overline{t}))]\!] \\ = \;& [\![\varphi(\mathsf{norm}(\overline{t})_{/\leftrightarrows})]\!]\,[\![\beta(\mathsf{norm}(\overline{t})_{/\leftrightarrows})]\!] \\ = \;& [\![\varphi(\mathsf{norm}(\overline{t}'))]\!]\,[\![\beta(\mathsf{norm}(\overline{t}'))]\!] \end{aligned}$$

But this necessitates our desired conclusion:

$$[\![\varphi(t)]\!]\,[\![\beta(t)]\!] = [\![\varphi(t')]\!]\,[\![\beta(t')]\!]$$

□

**Example 3.38.** Consider the bisimilar BESsy structure graphs from Example 3.36. We obtain the following equation systems for $\beta(t)$ and $\beta(t')$, respectively.

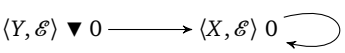
$$\begin{array}{ll} \nu X_w = X_w & \nu X_{w'} = X_{w'} \\ \mu X_t = X_v \vee X_w & \mu X_{t'} = X_{t'} \vee X_{w'} \\ \mu X_v = X_t \vee X_w & \end{array}$$

Observe that in both equation systems all variables are $\mathsf{true}$. The formulae are $\varphi(t) = X_v \vee X_w$ and $\varphi(t') = X_{t'} \vee X_{w'}$, and $[\![X_v \vee X_w]\!]\,[\![\beta(t)]\!] = [\![X_{t'}]\!]\,[\![\beta(t')]\!] = \mathsf{true}$.

## 3.4 Idempotence Identifying Bisimilarity of Structure Graphs

In the previous section we have seen that bisimulation respects a weak form of idempotence. The notion of *idempotence identifying bisimulation* was studied for equation systems in SRF in [KW11]. In *ibid.* it was shown that variables in such equation systems may be related if all variables in their right hand sides are related, regardless of the Boolean operands in these right hand sides. In this section we generalise idempotence identifying bisimulation to the setting of structure graphs, and we study the extent to which it is able to relate idempotent formulae. The following example shows that, indeed, strong bisimulation is not capable of relating idempotent formulae in situations where we desire to do so.

**Example 3.39.** Consider the structure graph corresponding to the equation system $\mathscr{E}$, defined as $(\nu Y = X \vee X)(\nu X = X)$, depicted below. Observe that the vertex for $Y$ is decorated by $\blacktriangledown$, and the vertex for $X$ is not decorated, therefore it is minimal modulo strong bisimulation.




Intuitively, we could simplify the right hand side of $Y$ to $X$ due to idempotence of $\vee$.

For equation systems in SRF this was resolved by the introduction of idempotence identifying bisimilarity in [KW11]. This is generalised to structure graphs as follows.

**Definition 3.40.** Let $\mathscr{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be a structure graph. A symmetric relation $R \subseteq T \times T$ is an idempotence identifying bisimulation relation if for all $(u, u') \in R$

- $r(u) = r(u')$, $\nearrow(u) = \nearrow(u')$;
- $d(u) \in \{\top, \bot\} \implies d(u) = d(u')$;
- $d(u) \neq d(u')$ implies for all $v, v' \in T$ if $u \rightarrow v$ and $u' \rightarrow v'$, then $(v, v') \in R$;
- for all $v \in T$, if $u \rightarrow v$, then $u' \rightarrow v'$ for some $v' \in T$ such that $(v, v') \in R$.

Two vertices $u$ and $u'$ are idempotence identifying bisimilar, notation $u \leftrightarrows u'$ if there exists an idempotence identifying bisimulation relation $R$ such that $(u, u') \in R$.

Observe that in the structure graph of Example 3.39 the vertices for $X$ and $Y$ are indeed idempotence identifying bisimilar.

### 3.4.1 Properties of Idempotence Identifying Bisimilarity

A *maximal* idempotence identifying bisimulation relation exists. This relation is the union of all possible idempotence identifying bisimulation relations.

*Property* 3.41. Let $R, S$ be two idempotence identifying bisimulation relations over normalised structure graphs. The union $R \cup S$ is again an idempotence identifying bisimulation relation.

We next show that idempotence identifying bisimilarity is an equivalence relation. We also show that it only has a natural quotienting operation if we restrict ourselves to normalised structure graphs.

**Proposition 3.42.** *idempotence identifying bisimilarity is an equivalence relation on structure graphs.*

*Proof.* Reflexivity and symmetry follow immediately. We therefore focus on transitivity. Assume that $u \leftrightarrows v$ and $v \leftrightarrows w$ for some $u, v$ and $w$. This means that there are idempotence identifying bisimulation relations $R$ and $S$ such that $u\ R\ v$ and $v\ S\ w$. Assume that $R$ and $S$ are such. Without loss of generality, assume that these relations are maximal. Then, (i) $u\ S \circ R\ w$, which follows by definition of $S \circ R$ (*viz.*, applying relation $S$ after $R$), and, (ii) $S \circ R$ is an idempotence identifying bisimulation relation, which we prove next.

Assume that $u\ S \circ R\ w$ for some $u, w$. Note that this means there has to be some $v$, such that $u\ R\ v$ and $v\ S\ w$. We show that $S \circ R$ satisfies the idempotence identifying bisimulation conditions.

- *ad* $r(u) = r(w)$. Observe that we have $u\ R\ v$ and $v\ S\ w$. From this, both $r(u) = r(v)$ and $r(v) = r(w)$ follow, proving $r(u) = r(w)$;
- *ad* $\nearrow u = \nearrow w$. Analogous to the previous case;
- *ad* $d(u) \in \{\top, \bot\} \implies d(u) = d(w)$. Observe that we have $u\ R\ v$ and $v\ S\ w$. From this, it follows that $d(u) \in \{\top, \bot\} \iff d(v) \in \{\top, \bot\}$, $d(u) \in \{\top, \bot\} \implies d(u) = d(v)$ and $d(v) \in \{\top, \bot\} \implies d(v) = d(w)$. From this it follows that $d(u) \in \{\top, \bot\} \implies (d(u) = d(v) \wedge d(v) = d(w))$, proving $d(u) \in \{\top, \bot\} \implies (d(u) = d(w))$ by transitivity of $=$;
- *ad* $d(u) \neq d(w)$ implies for all $u', w'$ if $u \to u'$ and $w \to w'$, then $(u', w') \in S \circ R$. Assume that $d(u) \neq d(w)$. Observe that $d(u), d(w)$, if defined, are not elements of $\{\top, \bot\}$. Towards a contradiction, assume there are $u', u''$ such that $u \to u'$, $u \to u''$ and $u' \not R\ u''$. Due to maximality of $R$, there are $v', v''$ such that

  $$v \to v', v \to v' \text{ and } v' \not R\ v''. \tag{3.1}$$

  Due to maximality of $S$, there also exist $v', v''$ such that $v \to v'$ and $v \to v''$ for which $v' \not S\ v''$. Likewise we derive that there exist $w', w''$ such that $w \to w'$ and $w \to w''$, for which $w' \not S\ w''$. From (3.1) and the fact that $R$ is an idempotence identifying bisimulation relation, the requirements from Definition 3.40 necessitate that $d(u) = d(v)$, and likewise $d(v) = d(w)$. This contradicts our assumption that $d(u) \neq d(w)$, hence for all $u', u''$ such that $u \to u'$ and $u \to u''$, we have that $u' R u''$. Then also for all $v', v''$ for which $v \to v'$ and $v \to v''$, $v' R v''$. Therefore, all $u'$ such that $u \to u'$ are related to all $w'$ for which $w \to w'$, via $S \circ R$;
- *ad* for all $u'$ s.t. $u \to u'$, there is a $w'$ with $w \to w'$, such that $u'\ S \circ R\ w'$. Let $u'$ be such that $u \to u'$. Since $R$ is an idempotence identifying bisimulation relation, there is some $v'$, $v \to v'$, such that $u'\ R\ v'$. Likewise, for this $v'$ there is some $w'$, $w \to w'$, such that $v'\ S\ w'$, hence $u'\ S \circ R\ w'$. □

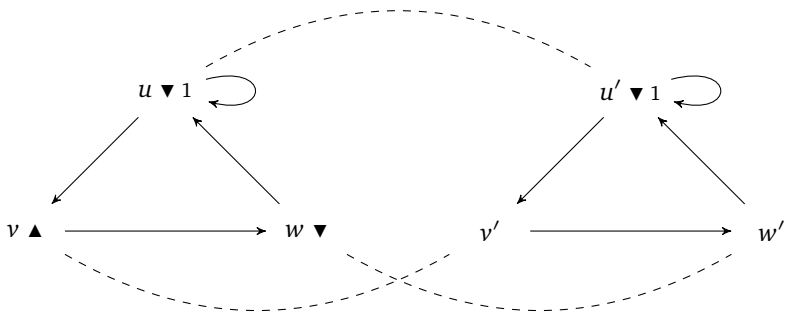
Using this notion of idempotence identifying bisimilarity, we also define the *idempotence identifying bisimulation quotient* of a structure graph.

**Definition 3.43.** Let $\mathcal{G} = \langle T, t, \rightarrow, d, r, \nearrow \rangle$ be a structure graph. The idempotence identifying bisimulation quotient $\mathcal{G}_{/\leftrightarrows} = \langle T', t', \rightarrow', d', r', \nearrow' \rangle$ of $\mathcal{G}$ is defined as follows:

- $T' \triangleq T_{/\leftrightarrows} = \{[t_i]_{\leftrightarrows} \mid t_i \in T\}$ with $[t_i]_{\leftrightarrows} = \{t_j \in T \mid t_i \leftrightarrows t_j\}$;
- $t' \triangleq [t]_{\leftrightarrows}$;
- $\rightarrow' \triangleq \{[t_i]_{\leftrightarrows} \rightarrow' [t_j]_{\leftrightarrows} \mid t_i \rightarrow t_j\}$;
- $d'([t_i]_{\leftrightarrows}) \triangleq \begin{cases} d(t_i) & \text{if } t_i \in \mathsf{dom}(d) \text{ and } |\{[t_j]_{\leftrightarrows} \mid [t_i]_{\leftrightarrows} \rightarrow' [t_j]_{\leftrightarrows}\}| \neq 1 \\ d(t_i) & \text{if } d(t_i) \in \{\top, \bot\} \\ \text{undefined} & \text{otherwise;} \end{cases}$
- $r'([t_i]_{\leftrightarrows}) \triangleq r(t_i)$, if $t_i \in \mathsf{dom}(r)$, and undefined otherwise;
- $\nearrow' ([t_i]_{\leftrightarrows}) \triangleq \nearrow (t_i)$, if $t_i \in \mathsf{dom}(\nearrow)$, and undefined otherwise.

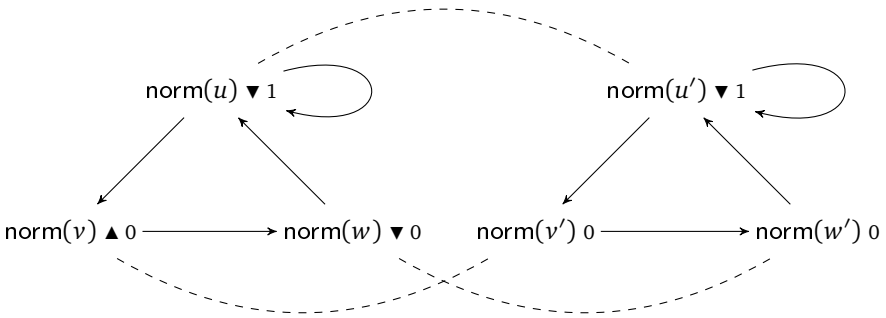
We showed that the strong bisimulation quotient of a BESsy structure graph is again BESsy in Lemma 3.21. The following example illustrates that this result does not hold for the idempotence identifying bisimulation quotient.

**Example 3.44.** Consider the following equation system $\mathcal{E}$, defined as $\mu X = X \vee ((X \vee X) \wedge (X \vee X))$, with the associated structure graph for $\langle X, \mathcal{E} \rangle$, represented by $u$, in the left structure graph below. Observe that this structure graph is BESsy, and that it is minimal modulo strong bisimulation.




The structure graph depicted to the right is the idempotence identifying bisimulation quotient of the graph to the left. The relation is depicted using dashed lines. Observe that the structure graph to the right is not BESsy, since it contains vertices with an outgoing edge that are not decorated with ▲, ▼ or a rank. Essentially, idempotence identifying bisimulation allows the removal of idempotence from unranked nodes in the structure graph, this removes all decorations from these nodes, whereas they retain at least one outgoing edge.

If, instead, we consider the normalised structure graph, we ensure that every vertex is ranked. In the following figure, we present the normalised structure graph corresponding to the previous BES. Its idempotence identifying bisimilarity quotient is shown to the right. The corresponding relation is again indicated.




Since the minimal idempotence identifying bisimilar structure graph that corresponds to the normalised BES is again BESsy, we can extract the following BES using $\beta(u')$:

$$\begin{aligned}
\mu X_{\mathsf{norm}(u')} &= X_{\mathsf{norm}(u')} \vee (X_{\mathsf{norm}(v')} \vee X_{\mathsf{norm}(v')}) \\
\nu X_{\mathsf{norm}(v')} &= X_{\mathsf{norm}(w')} \\
\nu X_{\mathsf{norm}(w')} &= X_{\mathsf{norm}(u')}
\end{aligned}$$

This gives rise to the following property of the idempotence identifying bisimilarity quotient, which is more restrictive than for strong bisimulation.

**Lemma 3.45.** *Let $\mathcal{G}$ be a normalised structure graph, then $\mathcal{G}_{/\leftrightarrows}$ is BESsy if $\mathcal{G}$ is BESsy.*

*Proof.* Let $\mathcal{G}$ be a normalised structure graph. Observe that in $\mathcal{G}$ every vertex is labelled by a rank or a free variable marking, and both are preserved in $\mathcal{G}_{/\leftrightarrows}$, satisfying the second and fourth requirement of BESSyness. If, in the quotient, a decoration ▲ or ▼ of a vertex $u$ is removed, this means that all successors of $u$ are related, hence in the quotient, the vertex corresponding to $u$ has only one successor, satisfying the third requirement of BESSyness. □

Observe that the converse of this lemma does not hold, *i.e.*, a minimal normalised BESsy structure graph can be equivalent to structure graphs that are not BESsy. This is witnessed by the structure graphs in Example 3.22 if we remove the decoration from vertex $\mathsf{norm}(w')$.

For Boolean equation systems in SRF, idempotence identifying bisimilarity was shown to relate equations of which the right hand sides are equivalent modulo idempotence. We first prove that this is still the case if we consider structure graphs.

**Lemma 3.46.** *Let $\mathscr{E}$ be a BES in SRF, such that $\sigma X = f \wedge f$ and $\sigma Y = f$ are equations in $\mathscr{E}$ satisfying $\mathsf{rank}_{\mathscr{E}}(X) = \mathsf{rank}_{\mathscr{E}}(Y)$. Then we have $\langle X, \mathscr{E}\rangle \leftrightarrows \langle Y, \mathscr{E}\rangle$.*

*Proof.* Suppose that $\mathsf{rank}_{\mathscr{E}}(X) = n$. Since $\mathscr{E}$ is in SRF, either $f$ is conjunctive, or $f$ is a variable. We distinguish both cases.

- $f$ is conjunctive. Then the vertices $\langle f \wedge f, \mathscr{E}\rangle$ and $\langle f, \mathscr{E}\rangle$ are both decorated by ▲ according to rule (5). Furthermore, both are not ranked. According to rule (19), the vertices $\langle X, \mathscr{E}\rangle$ and $\langle Y, \mathscr{E}\rangle$ are both labelled by ▲, and they are labelled by $\Uparrow n$ according to (2). Furthermore, since $f$ is conjunctive, $\langle f \wedge f, \mathscr{E}\rangle \to \langle g, \mathscr{E}\rangle$ if and only if $\langle f, \mathscr{E}\rangle \to \langle g, \mathscr{E}\rangle$, and all edges are inherited by the vertices for $X$ and $Y$ according to rule (23). Therefore, the identity relation extended with the pair $(\langle X, \mathscr{E}\rangle, \langle Y, \mathscr{E}\rangle)$ is an idempotence identifying bisimulation relation.

- $f$ is a variable. Observe that either $\langle f, \mathscr{E}\rangle \Uparrow m$ for some $m$, or $\langle f, \mathscr{E}\rangle \nearrow_Z$ for some $Z$. According to rules (21) and (22), $\langle Y, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$, and $\langle Y, \mathscr{E}\rangle$ is not decorated by ▲ or ▼. We also find that $\langle X, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$, because $\langle f \wedge f, \mathscr{E}\rangle \to \langle f, \mathscr{E}\rangle$, and the fact that $\langle f \wedge f, \mathscr{E}\rangle$ is decorated by ▲ and not ranked. Furthermore, $\langle X, \mathscr{E}\rangle$ is decorated by ▲ according to (19). Again, the identity relation extended with the pair $(\langle X, \mathscr{E}\rangle, \langle Y, \mathscr{E}\rangle)$ is an idempotence identifying bisimulation relation.

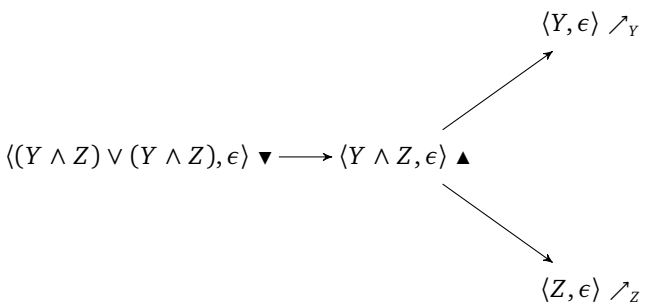
In both cases, we find that $\langle X, \mathscr{E}\rangle \leftrightarrows \langle f, Y\rangle$, hence the result follows. □

The following lemma, showing idempotence for disjunctive right hand sides in equation systems in SRF is symmetric.

**Lemma 3.47.** *Let $\mathscr{E}$ be a BES in SRF, such that $\sigma X = f \vee f$ and $\sigma Y = f$ are equations in $\mathscr{E}$ satisfying $\mathsf{rank}_{\mathscr{E}}(X) = \mathsf{rank}_{\mathscr{E}}(Y)$. Then we have $\langle X, \mathscr{E}\rangle \leftrightarrows \langle Y, \mathscr{E}\rangle$.*

The previous lemmata show that earlier results directly on equation systems in SRF carry over to the corresponding structure graphs. The question remains whether we can relate all idempotent formulae, *i.e.*, given an arbitrary equation system $\mathscr{E}$, and a formula $f$, is it the case that $\langle f \wedge f, \mathscr{E}\rangle \leftrightarrows \langle f, \mathscr{E}\rangle \leftrightarrows \langle f \vee f, \mathscr{E}\rangle$? The following example shows that this is, in general not the case.

**Example 3.48.** Consider the formulae $(Y \wedge Z) \vee (Y \wedge Z)$ an $(Y \wedge Z)$ in the empty equation system. This gives rise to the following structure graph.




There is no idempotence identifying bisimulation relation relating the vertices for $\langle (Y \wedge Z) \vee (Y \wedge Z), \epsilon\rangle$ and $\langle Y \wedge Z, \epsilon\rangle$, due to the sensitivity of idempotence identifying bisimilarity to counting.

In essence, this example shows that for two vertices in the structure graph to be related, the number of intermediate vertices on paths to vertices labelled $\nearrow$ or $\Uparrow$ must

be the same. Generally, this is not satisfied if we try to relate vertices that represent formulae with idempotent right hand sides in which Boolean connectives occur mixed. In particular, if we consider the formulae $X$ and $X \wedge X$ in the empty equation system, we do not have $\langle X, \epsilon \rangle \leftrightarrows \langle X \wedge X, \epsilon \rangle$. This follows immediately from the following structure graph. Similar examples using ranked vertices are as easy to construct.

$$\langle X \wedge X, \epsilon \rangle \blacktriangle \longrightarrow \langle X, \epsilon \rangle \nearrow_X$$

We already observed in Example 3.44 that quotienting is only well-defined for normalised structure graphs. The combination of Lemma 3.46 and the previous examples shows that for structure graphs, in general, idempotence identifying bisimulation does not live up to its name. This answers the question that we set out with: the extent to which idempotence identifying bisimilarity is able to relate vertices representing idempotent formulae is extremely limited in arbitrary structure graphs, but it is able to relate vertices representing equations with idempotent right hand sides in equation systems in SRF.

### 3.4.2 Idempotence Identifying Bisimilarity Implies Solution Equivalence

It remains to be shown that idempotence identifying bisimilarity preserves solution equivalence for normalised BESsy structure graphs. This is the first result of this section. In Theorem 3.51 we give an overview of the relationship between our equivalences.

We first show that normalised BESsy structure graphs without $\nearrow$ labelled vertices, that are idempotence identifying bisimilar, are solution equivalent.

**Lemma 3.49.** *Let $t, t'$ be normalised BESsy structure graphs in which no vertex is labelled with $\nearrow$. Assume $t$ is minimal with respect to idempotence identifying bisimilarity. Then $t \leftrightarrows t'$ implies $t \equiv t'$.*

*Proof.* Since no vertex is labelled with $\nearrow$, $\beta(t)$ and $\beta(t')$ are closed, and we need to show that $[\![\varphi(t)]\!][\![\beta(t)]\!] = [\![\varphi(t')]\!][\![\beta(t')]\!]$. Like in Lemma 3.35 we omit the case where the initial vertex of $t$ is marked with $\top$ or $\bot$, since it is trivial. Assume that the initial vertex of $t$ is not decorated with $\top$ nor $\bot$, and suppose that $[\![\varphi(t)]\!][\![\beta(t)]\!] = \mathsf{true}$. According to Proposition 3.32 we know that there is a $\blacktriangledown$-choice function $\gamma$ such that $[\![\beta(t_\gamma)]\!] = [\![\beta(t)]\!]$. As in the proof of Lemma 3.35, one can construct a $\blacktriangledown$-choice function $\gamma'$ for $t'$ satisfying:

$$\forall u \in \mathsf{dom}(\gamma), u' \in \mathsf{dom}(\gamma') : u \leftrightarrows u' \implies \gamma'(u) \leftrightarrows \gamma(u')$$

Note that a vertex $u'$ decorated with $\blacktriangle$ or $\blacktriangledown$ in $t'$ may be related to an undecorated vertex in $u$ in $t$ with $u \to v$, in which case an arbitrary $v'$ such that $u' \to v'$ may be chosen such that $u' \leftrightarrows v'$. The proof that $t_\gamma$ and $t'_{\gamma'}$ have the same paths follows the same line of reasoning as in the proof of Lemma 3.35, and results in $[\![\varphi(t')]\!][\![\beta(t')]\!] = \mathsf{true}$, hence $t \equiv t'$. The case $[\![\varphi(t)]\!][\![\beta(t)]\!] = \mathsf{false}$ is again fully dual. □

The next theorem generalises this result to structure graphs with vertices labelled $\nearrow$. Due to Example 3.44 we only consider normalised BESsy structure graphs, whereas Theorem 3.37 reasoned about arbitrary BESsy structure graphs. The intent of both theorems is similar, however.

**Theorem 3.50.** *Let $t$, $t'$ be BESsy structure graphs, then $\mathsf{norm}(t) \leftrightarrows \mathsf{norm}(t')$ implies $t \equiv t'$.*

*Proof.* Follows the exact same line of reasoning as the proof of Theorem 3.37, using Lemma 3.49. □

We can now summarise the relationships between the equivalences for normalised, BESsy structure graphs.

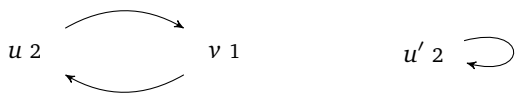
**Theorem 3.51.** *On normalised, BESsy structure graphs, we have:*

1. *the relation $\leftrightarrows$ is strictly finer than $\leftrightarrows$;*
2. *the relation $\leftrightarrows$ is strictly finer than $\equiv$.*

*Proof.* We show each property separately.

1. $ad \leftrightarrows \subseteq \leftrightarrows$; observe that $\leftrightarrows$ is finer than $\leftrightarrows$ as an immediate consequence of the weakening of the first requirement of bisimilarity: for all bisimilar vertices $u$, $u'$ having the same decorations, *i.e.*, $d(u) = d(u')$, the third requirement of idempotence identifying is trivially satisfied. Strictness follows from the fact that in the following two structure graphs $u$ and $u'$ are idempotence identifying bisimilar, but not bisimilar.


2. $ad \leftrightarrows \subseteq \equiv$; follows immediately from $t \leftrightarrows t_{/\leftrightarrows}$ for all $t$, the fact that $\leftrightarrows$ and $\equiv$ are equivalence relations, and Theorem 3.50. Strictness follows from the fact that in the following two structure graphs $u$ and $u'$ are solution equivalent, but not idempotence identifying bisimilar.




□

The following proposition demonstrates that using idempotence identifying bisimilarity subgraphs sometimes reduce to a single vertex. In essence, this shows that for this class of structure graphs, idempotence identifying bisimilarity computes the solution.

**Proposition 3.52.** *Let $t$ be normalised, BESsy structure graph, in which no vertex is labelled $\nearrow$, $\top$ or $\bot$. Let $u$ be a subgraph of $t$, that is closed with respect to $\rightarrow$, and in which all vertices have the same rank. Then for all $v, w$ in $u$, $v \leftrightarrows w$ and $v \equiv w$.*

*Proof.* The relation $R$ relating all vertices in $u$ is an idempotence identifying bisimulation. Observe that, in such a graph $u$ all paths are infinite, and dominated by a single priority, hence, according to Lemma 3.34, the formulae corresponding to the vertices in $u$ all have the same solution. From these two observations the result follows immediately. □

In other words, a subgraph without outgoing edges, consisting of vertices all labelled with the same rank, and without decorations $\nearrow$, $\top$ or $\bot$, can be reduced to a single vertex. The following example shows that idempotence identifying bisimilarity can yield a substantially greater reduction, by an arbitrarily large factor, than bisimilarity.

**Example 3.53.** Consider the following class of structure graphs in which we have a sequence of vertices with alternating decorations ▼ and ▲ of arbitrary length. Due the presence of the decorations, every structure graph in this class is minimal with respect to bisimulation. Under idempotence identifying bisimulation every structure graph in this class reduces to a single vertex with priority 0 and a self-loop.


This structure graph (modulo naming of the vertices) can, *e.g.*, be obtained from the BES $(\nu X_1 = X_2 \vee X_2)(\nu X_2 = X_3 \wedge X_3) \cdots (\nu X_{n-1} = X_n \wedge X_n)(\nu X_n = X_n)$

## 3.5 Bisimilarity on Processes vs Bisimilarity on Structure Graphs

The $\mu$-calculus and bisimilarity of finitely branching labelled transition systems are intimately related: two states in a transition system are bisimilar if and only if the states satisfy the same set of $\mu$-calculus formulae. As a result, one can rely on bisimulation minimisation techniques for reducing the complexity of the labelled transition system, prior to analysing whether a given $\mu$-calculus formula holds for that system. Unfortunately, in practice, bisimulation reductions are often disappointing, and have to be combined with abstractions that are safe with respect to the formula in order to be worthwhile.

We show that minimising an equation system that encodes a model checking problem is, size-wise, always at least as effective as first applying a safe abstraction to the labelled transition system, subsequently minimising the latter and only then encoding the model checking problem in an equation system. An additional example illustrates that bisimulation minimisation for equation systems can in fact be more effective.

We first prove that, if for all bisimilar states $s, s'$ in a labelled transition system, the corresponding $X_s$ and $X_{s'}$ are also bisimilar, that then the vertices representing the right hand sides of $s$ and $s'$ in the structure graph are also bisimilar. Note that we consider arbitrary equation systems. In Proposition 3.55 we use this lemma to prove that bisimilar states in a labelled transition system give rise to bisimilar vertices in the structure graph corresponding to the equation system encoding the model checking problem on the LTS.

**Lemma 3.54.** *Assume $L = \langle S, Act, \rightarrow \rangle$ is an arbitrary labelled transition system. Let $\phi$ be an arbitrary $\mu$-calculus formula. Then, for arbitrary equation systems $\mathscr{E}$, we have:*

$$\begin{array}{rl} \textit{if} & \forall s, s' \in S : s \leftrightarroweq s' \implies \forall \tilde{X} \in \mathsf{bnd}(\phi) \cup \mathsf{occ}(\phi) : \langle X_s, \mathscr{E} \rangle \leftrightarroweq \langle X_{s'}, \mathscr{E} \rangle \\ \textit{then} & \forall s, s' \in S : s \leftrightarroweq s' \implies \langle \mathsf{RHS}_s(\phi), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{s'}(\phi), \mathscr{E} \rangle \end{array}$$

*Proof.* Assume a given equation system $\mathscr{E}$. We proceed by means of an induction on the structure of $\phi$.

- *Base cases.* Assume that for all $s, s' \in S$, satisfying $s \leftrightarroweq s'$, and all $\tilde{X} \in \mathsf{bnd}(\phi) \cup \mathsf{occ}(\phi)$, we have $\langle X_s, \mathscr{E} \rangle \leftrightarroweq \langle X_{s'}, \mathscr{E} \rangle$. Assume that $t, t' \in S$ are arbitrary states satisfying $t \leftrightarroweq t'$.
  - ad $\phi \equiv b$, where $b \in \{\mathsf{true}, \mathsf{false}\}$. Clearly, since $\langle \mathsf{RHS}_t(\phi), \mathscr{E} \rangle = \langle b, \mathscr{E} \rangle = \langle \mathsf{RHS}_{t'}(\phi), \mathscr{E} \rangle$, bisimilarity is guaranteed by unicity of the term, regardless of the states $t$ and $t'$;
  - ad $\phi \equiv \tilde{X}$. Clearly, $\tilde{X} \in \mathsf{occ}(\phi)$, so, the required conclusion follows immediately from the fact that $\langle \mathsf{RHS}_t(\phi), \mathscr{E} \rangle = \langle X_t, \mathscr{E} \rangle \leftrightarroweq \langle X_{t'}, \mathscr{E} \rangle = \langle \mathsf{RHS}_{t'}(\phi), \mathscr{E} \rangle$;
- Inductive cases: we assume the following induction hypothesis:

$$\begin{array}{rl} \text{if} & \forall s, s' \in S : s \leftrightarroweq s' \implies \forall \tilde{X} \in \mathsf{bnd}(\phi_i) \cup \mathsf{occ}(\phi_i) : \langle X_s, \mathscr{E} \rangle \leftrightarroweq \langle X_{s'}, \mathscr{E} \rangle \\ \text{then} & \forall s, s' \in S : s \leftrightarroweq s' \implies \langle \mathsf{RHS}_s(\phi_i), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{s'}(\phi_i), \mathscr{E} \rangle \end{array} \tag{IH}$$

  From hereon, assume that we have a pair of bisimilar states $t, t' \in S$.
  - ad $\phi \equiv \phi_1 \wedge \phi_2$. Assume that for any pair of bisimilar states $s, s' \in S$, and for all $\tilde{X} \in \mathsf{bnd}(\phi_1 \wedge \phi_2) \cup \mathsf{occ}(\phi_1 \wedge \phi_2) = (\mathsf{bnd}(\phi_1) \cup \mathsf{occ}(\phi_1)) \cup (\mathsf{bnd}(\phi_2) \cup \mathsf{occ}(\phi_2))$, we have $\langle X_s, \mathscr{E} \rangle \leftrightarroweq \langle X_{s'}, \mathscr{E} \rangle$. By our induction hypothesis, we have $\langle \mathsf{RHS}_t(\phi_1), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{t'}(\phi_1), \mathscr{E} \rangle$ and $\langle \mathsf{RHS}_t(\phi_2), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{t'}(\phi_2), \mathscr{E} \rangle$. According to Lemma 3.23 we get $\langle \mathsf{RHS}_t(\phi_1) \wedge \mathsf{RHS}_t(\phi_2), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{t'}(\phi_1) \wedge \mathsf{RHS}_{t'}(\phi_2), \mathscr{E} \rangle$. By definition of $\mathsf{RHS}$, we have the required $\langle \mathsf{RHS}_t(\phi_1 \wedge \phi_2), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{t'}(\phi_1 \wedge \phi_2), \mathscr{E} \rangle$.
  - ad $\phi \equiv \phi_1 \vee \phi_2$. Follows the same line of reasoning as the previous case.
  - ad $\phi \equiv [A]\phi_1$. Assume that for all pairs of bisimilar states $s, s' \in S$, and all $\tilde{X} \in \mathsf{bnd}([A]\phi_1) \cup \mathsf{occ}([A]\phi_1) = \mathsf{bnd}(\phi_1) \cup \mathsf{occ}(\phi_1)$, we have $\langle X_s, \mathscr{E} \rangle \leftrightarroweq \langle X_{s'}, \mathscr{E} \rangle$. By induction, we find that $\langle \mathsf{RHS}_s(\phi_1), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{s'}(\phi_1), \mathscr{E} \rangle$ holds for all pairs of bisimilar states $s, s' \in S$. This includes states $t$ and $t'$. Since $t$ and $t'$ are bisimilar, we have $t \xrightarrow{a}$ if and only if $t' \xrightarrow{a}$ for all $a \in A$. We distinguish two cases:
    1. Case $t \not\xrightarrow{a}$ for any $a \in A$. Then also $t' \not\xrightarrow{a}$ for any $a \in A$. Hence, $\mathsf{RHS}_t([A]\phi_1) = \mathsf{true} = \mathsf{RHS}_{t'}([A]\phi_1)$. We thus immediately have the required $\langle \mathsf{RHS}_t([A]\phi_1), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{t'}([A]\phi_1), \mathscr{E} \rangle$;
    2. Case $t \xrightarrow{a}$ for some $a \in A$. Assume that $t \xrightarrow{a} u$. Since $t \leftrightarroweq t'$, we have $t' \xrightarrow{a} u'$ for some $u' \in S$ satisfying $u \leftrightarroweq u'$ (and *vice versa*). Because of our induction hypothesis, we then also have $\langle \mathsf{RHS}_u(\phi_1), \mathscr{E} \rangle \leftrightarroweq \langle \mathsf{RHS}_{u'}(\phi_1), \mathscr{E} \rangle$

(and *vice versa*). We thus find that for every term in the non-empty set $\{\langle \mathsf{RHS}_u(\phi_1), \mathscr{E}\rangle \mid a \in A, t \xrightarrow{a} u\}$, we can find a bisimilar term in the set $\{\langle \mathsf{RHS}_{u'}(\phi_1), \mathscr{E}\rangle \mid a \in A, t' \xrightarrow{a} u'\}$ and *vice versa*. Then, by Corollary 3.25, also $\langle \bigsqcap\{\mathsf{RHS}_u(\phi_1) \mid a \in A, t \xrightarrow{a} u\}, \mathscr{E}\rangle \leftrightarrows \langle \bigsqcap\{\mathsf{RHS}_{u'}(\phi_1) \mid a \in A, t' \xrightarrow{a} u'\}, \mathscr{E}\rangle$. This leads to $\langle \mathsf{RHS}_t([A]\phi_1), \mathscr{E}\rangle \leftrightarrows \langle \mathsf{RHS}_{t'}([A]\phi_1), \mathscr{E}\rangle$.

Clearly, both cases lead to the required conclusion.

- ad $\phi \equiv \langle A\rangle\phi_1$. Follows the same line of reasoning as the previous case.
- ad $\phi \equiv \sigma\tilde{X}.\ \phi_1$. Since $\tilde{X} \in \mathsf{bnd}(\phi)$, this case follows immediately from the assumption on $\tilde{X}$ and the definition of $\mathsf{RHS}$.

□

The above lemma is at the basis of the following proposition:

**Proposition 3.55.** *Let $L = \langle S, Act, \rightarrow\rangle$ be a labelled transition system. Let $\phi$ be an arbitrary closed $\mu$-calculus formula. Let $s, s' \in S$ be an arbitrary pair of bisimilar states. We then have:*

$$\forall \tilde{X} \in \mathsf{bnd}(\phi) : \langle X_s, \mathsf{E}^L(\phi)\rangle \leftrightarrows \langle X_{s'}, \mathsf{E}^L(\phi)\rangle$$

*Proof.* Let $\phi$ be an arbitrary closed formula, *i.e.*, $\mathsf{occ}(\phi) \subseteq \mathsf{bnd}(\phi)$; since $\phi$ is a closed formula, $\mathsf{E}^L(\phi)$ will be a closed equation system. In case $\mathsf{bnd}(\phi) = \emptyset$, the statement holds vacuously. Assume $\mathsf{bnd}(\phi) = \{\tilde{X}^1, \ldots, \tilde{X}^n\}$, for some $n \geq 1$. Clearly, for each variable $\tilde{X}^i \in \mathsf{bnd}(\phi)$, we obtain equations of the form $\sigma_i X^i_s = \mathsf{RHS}_s(f^i)$ in $\mathsf{E}^L(\phi)$. Let $I$ be the relation on vertices, defined as follows:

$$I = \{(\langle X^i_s, \mathsf{E}^L(\phi)\rangle, \langle X^i_{s'}, \mathsf{E}^L(\phi)\rangle) \mid s, s' \in S, \tilde{X}^i \in \mathsf{bnd}(\phi), s \leftrightarrows s'\}$$

According to Lemma 3.54, $I$ underlies the bisimilarity between $\langle \mathsf{RHS}_s(f^i), \mathsf{E}^L(\phi)\rangle$ and $\langle \mathsf{RHS}_{s'}(f^i), \mathsf{E}^L(\phi)\rangle$ for pairs of bisimilar states $s, s' \in S$. Assume $R_{f^i}$ is the bisimulation relation underlying said equivalence. Let $R$ be defined as follows:

$$R = I \cup \bigcup_{f^i} R_{f^i}$$

$R$ is again a bisimulation relation, as can be checked using the SOS rules 19–24 for equations and Lemma 3.54. Clearly, $R$ relates $\langle X_s, \mathsf{E}^L(\phi)\rangle$ and $\langle X_{s'}, \mathsf{E}^L(\phi)\rangle$ for arbitrary $\tilde{X} \in \mathsf{bnd}(\phi)$ and bisimilar states $s, s' \in S$. □

As a result of the above proposition one can argue that bisimulation on processes is less powerful than bisimulation on equation systems. However, one may be inclined to believe that combined with abstraction, bisimilarity on processes can lead to greater reductions. Below, we show that even in the presence of *safe* abstractions, bisimilarity on equation systems still surpasses bisimilarity on processes.

We first formalise the notion of safe abstraction for processes. Assume $\tau$ is a constant, not present in any set of actions *Act*.

**Definition 3.56.** An *abstraction* of a labelled transition system $L = \langle S, Act, \rightarrow\rangle$ with respect to a set of actions $A \subseteq Act$, is the labelled transition system $L_A = \langle S, Act \cup \{\tau\}, \rightarrow_A\rangle$, where:

- for all actions $a \notin A$, $s \xrightarrow{a}_A s'$ if and only if $s \xrightarrow{a} s'$;
- $s \xrightarrow{\tau}_A s'$ if and only if $s \xrightarrow{a} s'$ for some $a \in A$;

In effect, an abstraction relabels an action that decorates a transition to $\tau$ only if that action appears in the set $A$. Clearly, if $s \leftrightarrows s'$ holds in $L$, then also $s \leftrightarrows s'$ in $L_A$, but the converse does not hold necessarily.

**Definition 3.57.** An abstraction $L_A$ of $L$ is said to be *safe* with respect to a closed modal $\mu$-calculus formula $\phi$ if and only if for each subformula $[A']\psi$ and $\langle A'\rangle\psi$ of $\phi$, $A' \cap A = \emptyset$.

It follows from the semantics of the modal $\mu$-calculus that all actions of some $L$, disjoint with the actions found inside the modalities in $\phi$ can be renamed to $\tau$ without affecting the validity of the model checking problem.

**Proposition 3.58.** *Let $L = \langle S, Act, \rightarrow\rangle$ be a labelled transition system. Let $\phi$ be a closed modal $\mu$-calculus formula, and assume $L_A$ is a safe abstraction of $L$. Then for each state $s \in S$, we have $L, s \models \phi$ if and only if $L_A, s \models \phi$.*

The below theorem strengthens the result we obtained in Proposition 3.55, by stating that even in the presence of safe abstractions, bisimilarity for equation systems is as powerful as bisimilarity taking abstractions into account.

**Theorem 3.59.** *Let $L = \langle S, Act, \rightarrow\rangle$ be an arbitrary labelled transition system. Let $\phi$ be an arbitrary closed modal $\mu$-calculus formula over Act. Then for every safe abstraction $L_A$ of $L$, we have for every pair of bisimilar states $s, s' \in S$ in $L_A$:*

$$\forall X \in \mathsf{bnd}(\phi) : \langle X_s, \mathsf{E}^L(\phi)\rangle \leftrightarrows \langle X_{s'}, \mathsf{E}^L(\phi)\rangle$$

*Proof.* The proof is similar to the proof of Proposition 3.55. In particular, it relies on the definition of a safe abstraction to ensure that $\langle \mathsf{RHS}_s([A']\psi), \mathscr{E}\rangle$ and $\langle \mathsf{RHS}_{s'}([A']\psi), \mathscr{E}\rangle$ for states $s, s'$ that are bisimilar in $L_A$, but not in $L$, are mapped onto $\langle \mathsf{true}, \mathscr{E}\rangle$ for both LTSs. □

Theorem 3.59 positively shows that bisimilar states in a state space indeed give rise to bisimilar equations in the equation systems encoding model checking problems, even when considering 'safe' abstractions on the original state space.

Using Theorem 3.27 and Theorem 3.51 we immediately get the following as a corollary of Theorem 3.59.

**Corollary 3.60.** *Let $L = \langle S, Act, \rightarrow\rangle$ be an arbitrary labelled transition system. Let $\phi$ be an arbitrary closed $\mu$-calculus formula over Act. Then for every safe abstraction $L_A$, we have for every pair of bisimilar states $s, s' \in S$ in $L_A$:*

$$\forall \tilde{X} \in \mathsf{bnd}(\phi) : \mathsf{norm}(\langle X_s, \mathsf{E}^L(\phi)\rangle) \leftrightarrows \mathsf{norm}(\langle X_{s'}, \mathsf{E}^L(\phi)\rangle)$$

Lastly, we provide an example that demonstrates that bisimulation reduction of equation systems can lead to arbitrarily larger reductions compared to the reductions achievable through safe abstractions and minimisation of a given LTS.

**Example 3.61.** Let $N$ be an arbitrary positive number. Consider the process described by the following set of recursive processes (using process algebra style notation):

$$\{P_1 = a \cdot Q_N, \quad P_{n+1} = a \cdot P_n, \quad Q_1 = b \cdot P_N, \quad Q_{n+1} = b \cdot Q_n \mid n < N\}$$

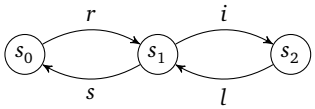
Process $P_N$ induces an LTS $L$ that performs a sequence of $a$ actions of length $N$, followed by a sequence of $b$ actions of length $N$, returning to process $P_N$. Observe that the process $P_N$ cannot be reduced further modulo bisimulation. Let $\phi$ be the modal $\mu$-calculus formula $\phi = \nu\tilde{X}.\ \langle\{a,b\}\rangle\tilde{X}$, asserting that there is an infinite sequence consisting of $a$'s, $b$'s, or $a$'s and $b$'s. Clearly, there is no safe abstraction of process $P_N$ with respect to $\phi$, other than process $P_N$ itself. The equation system $\mathsf{E}^{P_N}(\phi)$ is as follows:

$$\nu\{(X_{P_1} = X_{Q_N} \vee X_{Q_N}), (X_{P_{n+1}} = X_{P_n} \vee X_{P_n}),\\ (X_{Q_1} = X_{P_N} \vee X_{P_N}), (X_{Q_{n+1}} = X_{Q_n} \vee X_{Q_n}) \mid n < N\}$$

We find that $\langle X_{P_N}, \mathsf{E}^{P_N}(\phi)\rangle$ and $\langle Y, (\nu Y = Y \vee Y)\rangle$ are bisimilar, which demonstrates a reduction of a factor $2N$. As the labelled transition system can be scaled to arbitrary size, this demonstrates that bisimilarity for equation systems can be arbitrarily more effective.

## 3.6 Application

Equation systems that are not immediately in simple form can be obtained through the reduction of process equivalence checking problems such as the branching bisimulation problem, see *e.g.* [Che+07], and the more involved model checking problems. As a slightly simplified example of the latter, we analyse an unreliable channel using $\mu$-calculus model checking. The channel can read messages from its environment through the $r$ action, and send or lose these next through the $s$ action and the $l$ action, respectively. Losing a message happens because of noise affecting the reliability of the channel; we model this using an internal action $i$ preceding action $l$. In case the message is lost, subsequent attempts are made to send the message until this finally succeeds. The labelled transition system, modelling this system is given below.




Suppose we wish to verify for which states it holds whether along all paths consisting of reading and sending actions, it is infinitely often possible to potentially never perform a send action. Intuitively, this should be the case in all states: from states $s_0$ and $s_1$, there is a finite path leading to state $s_1$, which can subsequently produce the infinite path $(s_1\ s_2)^\omega$, along which the send action does not occur. For state $s_2$, we observe that there is no path consisting of reading and sending actions, so the property holds vacuously in $s_2$. We formalise this problem as follows:[1]

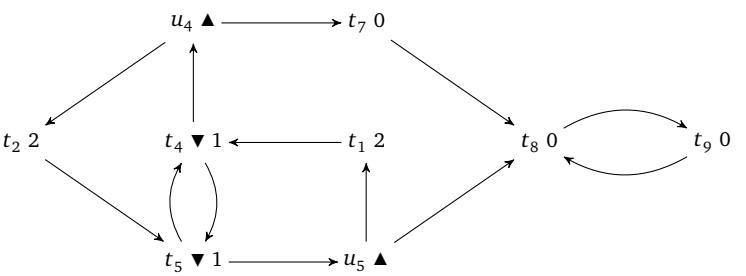
$$\phi \equiv \nu\tilde{X}.\ \mu\tilde{Y}.\ (([\{r,s\}]\tilde{X} \wedge (\nu\tilde{Z}.\ \langle\bar{s}\rangle\tilde{Z})) \vee [\{r,s\}]\tilde{Y})$$

[1] Alternative phrasings are possible, but this one nicely projects onto an equation system with non-trivial right-hand sides, clearly illustrating the theory outlined in the previous sections in an example of manageable proportions.

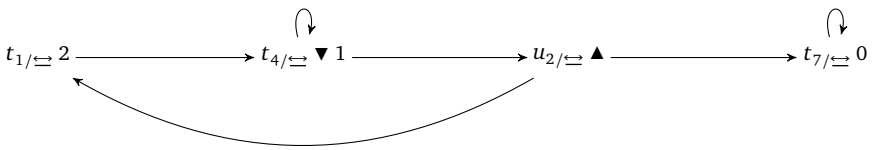
Verifying which states in the labelled transition system satisfy $\phi$ is answered by solving the equation system below. Note that the equation system was obtained through Definition 2.35. The solution to $X_{s_i}$ answers whether $s_i \models \phi$.

$$
\begin{aligned}
&(\nu X_{s_0} = Y_{s_0})\\
&(\nu X_{s_1} = Y_{s_1})\\
&(\nu X_{s_2} = Y_{s_2})\\
&(\mu Y_{s_0} = ((X_{s_1} \wedge X_{s_1}) \wedge Z_{s_0}) \vee ((Y_{s_1} \wedge Y_{s_1}) \vee (Y_{s_1} \wedge Y_{s_1})))\\
&(\mu Y_{s_1} = ((X_{s_0} \wedge X_{s_0}) \wedge Z_{s_1}) \vee ((Y_{s_0} \wedge Y_{s_0}) \vee (Y_{s_0} \wedge Y_{s_0})))\\
&(\mu Y_{s_2} = (\mathsf{true} \wedge Z_{s_2}) \vee \mathsf{true})\\
&(\nu Z_{s_0} = Z_{s_1} \vee Z_{s_1})\\
&(\nu Z_{s_1} = Z_{s_2} \vee Z_{s_2})\\
&(\nu Z_{s_2} = Z_{s_1} \vee Z_{s_1})
\end{aligned}
$$

An answer to the global model checking problem would be encoded by the structure graph $\langle X_{s_0} \wedge X_{s_1} \wedge X_{s_1}, \mathsf{E}^L(\phi)\rangle$. Here we only depict the structure graph encoding the local model checking problem $s_0 \models \phi$, encoded by the structure graph $\langle X_{s_0}, \mathsf{E}^L(\phi)\rangle$, which has initial vertex $t_1$. Note that the ranked vertices $t_i$ originate from the $i^{th}$ equation in the equation system. Likewise, the unranked vertices $u_i$ originate from the right-hand side of the $i^{th}$ equation.




Observe that we have $t_1 \leftrightarrows t_2$, $t_7 \leftrightarrows t_8 \leftrightarrows t_9$, $t_4 \leftrightarrows t_5$ and $u_4 \leftrightarrows u_5$. Minimising the above structure graph with respect to bisimulation leads to the structure graph depicted below:




Note that the structure graph is BESsy, and, hence, admits a translation back to an equation system. Using the translation provided in Definition 3.8 results in the following equation system:

$$
\begin{aligned}
&(\nu X_{t_{1/\leftrightarrows}} = X_{t_{4/\leftrightarrows}})\\
&(\mu X_{t_{4/\leftrightarrows}} = (X_{t_{7/\leftrightarrows}} \wedge (X_{t_{1/\leftrightarrows}} \wedge X_{t_{1/\leftrightarrows}})) \vee (X_{t_{4/\leftrightarrows}} \vee X_{t_{4/\leftrightarrows}}))\\
&(\nu X_{t_{7/\leftrightarrows}} = X_{t_{7/\leftrightarrows}})
\end{aligned}
$$

Answering the verification problem $s_0 \models \phi$ can thus be achieved by solving 3 equations rather than the original 9 equations. Using standard algorithms for solving equation systems, one quickly finds that all equations of the minimised equation system (and thereby all of the equations from the original equation system they represent) have true as their solutions. Note that the respective sizes of the structure graphs underlying the required equations in the original equation systems are 9 before minimisation and 4 after minimisation, which is almost a 55% gain. Such gains (and larger) appear to be typical in this setting (see also [KW11]), and often surpass those in the setting of labelled transition systems. Similar gains are found for the global model checking problem. Observe, moreover, that the original labelled transition system already is minimal, demonstrating once more that the minimisation of an equation system can be more effective than minimising the original labelled transition system.

## 3.7 Closing Remarks

We presented a set of deduction rules for deriving *structure graphs* from proposition formulae and Boolean equation systems, following the regime of [Plo04]. In defining these rules, we focussed on simplicity. We carefully selected a small set of computationally cheap logical equivalences that we wished to be reflected by bisimilarity in our structure graphs, and subsequently showed that we met these goals.

Structure graphs generalise the *dependency graphs* of *e.g.* [Mad97; Kei06]. The latter formalism is incapable of capturing all the syntactic riches of Boolean equation systems, and is only suited for a subset of closed equation systems in simple form. A question, put forward in [KW11], is how the restriction to equation systems in simple form affects the power of reduction of strong bisimulation. In Section 3.3, we showed that these restrictions are in fact beneficial to the identifying power of bisimilarity. This result follows immediately from the meta-theory for structured operational rules, see *e.g.* [MRG05]. We furthermore proved that also in our richer setting, bisimulation minimisation of a structure graph, induced by an equation system, preserves and reflects the solution to the original equation system. This generalises [KW11, Theorem 1] for dependency graphs. In Section 3.4 we showed that, by weakening the definition of bisimulation, we can equate vertices representing conjunctive and disjunctive formulae. However, this idempotence identifying bisimilarity is only meaningful on a subset of the structure graphs. This may be a fair price to pay, since for some subclasses of structure graphs the reduction achieved using idempotence identifying bisimilarity is arbitrarily larger than that achieved using strong bisimilarity.

Beyond the aforementioned results, we studied the connection between bisimilarity for labelled transition systems, the $\mu$-calculus model checking problem and (idempotence identifying) bisimilarity for structure graphs. In Section 3.5, we showed that bisimulation minimisation of a structure graph (associated to an equation system encoding an arbitrary model checking problem on an arbitrary labelled transition system) is at least as effective as bisimulation minimisation of the labelled transition system prior to the encoding. This relation even holds when bisimilarity is combined with safe abstractions for labelled transition systems. We moreover show that this relation is strict through an example formula $\phi$ and a labelled transition system $L$ of $2N$ ($N \geq 1$) states that is already

minimal (even when considering safe abstractions with respect to $\phi$), whereas the structure graph induced by the equation system encoding the model checking problem can be reduced by a factor $2N$. These results provide the theoretical underpinning for the huge reductions observed in [KW11]. Results of similar experiments, applied to parity games instead of Boolean equation systems, are presented in Chapter 5. Reducing a labelled transition system (if available explicitly), prior to encoding the verification problem as a Boolean equation system, can still be useful, as the encoding is proportional in the size of the labelled transition system.

The structure graphs that we considered in this chapter are of both theoretical and practical significance. They generalise various graph-based models, including the aforementioned dependency graphs, but also Parity Games [Zie98], and there are strong links between our structure graphs and Switching Graphs [GP09] which have two kinds of edges: ordinary edges and *switches*, which can be set to one of two destinations. Switching Graphs are more general than the dependency graphs of [Kei06], but are still inadequate for directly capturing the structure of the entire class of equation systems. Note that in the Switching Graph setting, the *v-parity loop problem* is equivalent to the problem of solving Boolean equation systems.

Given these links, a *game-based* characterisation of the concept of solution for equation systems, stated in terms of our choice functions and structure graphs is open for investigation.

Simulation relations for Parity Games have been studied in, *e.g.*, [FW06]. Some of those are computationally expensive, or do not have well-defined quotients. In the next chapter, we study equivalences weaker than bisimilarity and idempotence identifying bisimilarity for parity games, striking a balance between expressivity and computational complexity. We refrain from presenting that theory in the more general framework of structure graph. We have shown that for bisimilarity, normalising the structure graph—effectively turning it into a parity game—is beneficial to the reduction, and for idempotence identifying bisimulation it is even essential. This suggests that weakening equivalences for parity games might be beneficial. Furthermore, the notational overhead of structure graphs turns out to be quite large; restricting ourselves to parity games allows us to focus on the essence of the problem.

# Chapter 4

# Equivalences on parity games

In the previous chapter we studied the notion of structure graph for Boolean equation systems. We observed that normalisation of the structure does not negatively influence the capabilities of bisimulation; in fact, bisimulation can reduce structure graphs by an arbitrarily larger factor.

Normalised BESsy structure graphs in which no vertex is labelled with $\top$, $\bot$ or $\nearrow$ directly induce Boolean equation systems in simple recursive form. These Boolean equation systems in turn have a direct one-to-one correspondence with parity games, as we showed in Chapter 2. Strong bisimulation ($\leftrightarrows$) and idempotence identifying bisimulation can easily be recast in terms of parity games. In this setting we use the term *governed bisimulation* ($\leftrightarrows$) instead of idempotence identifying bisimulation—the notion of idempotence does not exist in the context of parity games. These are the first two equivalences for parity games that we introduce.

In the literature, other equivalences and preorders for parity games have been described. The notion of *direct simulation* ($\equiv_{dir}$) for parity games is hinted at in, *e.g.*, [FW06]. Fritz and Wilke observed that, under some restrictions, we do not care whether taking a certain step is postponed a finite number of times—the winner of a play in a parity game only depends on the priorities that occur infinitely often, and these priorities are preserved. This observation is used in their definition of *delayed simulation* preorder and -equivalence ($\equiv_{de}$) [FW06]. Delayed simulation does not have a well-defined quotient. They introduced $\Diamond$-biased ($\equiv^{e}_{de}$) and $\Box$-biased ($\equiv^{o}_{de}$) variations of delayed simulation that do have a well-defined quotient.

Similar to the observation by Fritz and Wilke, we observe that finite stretches of vertices with the same priority on a play may be compressed into a single vertex. In this chapter we use this observation to define two additional equivalences on parity games.

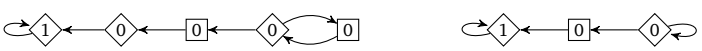
In defining our equivalences we draw inspiration from process theory. The notion of *stuttering equivalence* ($\simeq$), also referred to as stuttering bisimulation, is well-known for Kripke structures. Stuttering equivalence allows for compressing sequences of vertices with the same state label, as long as none of the possible future behaviours is removed. Furthermore, it only relates two vertices if either both have the possibility to *diverge*, *i.e.*, stay within the same equivalence class indefinitely, or both do not have this possibility.

If we assume that we may only relate vertices if they are owned by the same player

and they have the same priority, this idea carries over to parity games. We show that stuttering equivalent vertices are won by the same player in the parity game. As a side result, given a winning strategy for a player from a particular vertex, we obtain winning strategies for all stuttering equivalent vertices. This is of particular interest in case one is seeking an explanation for the solution of the game, for instance as a means of diagnosing a failed verification.

Stuttering equivalence enables us to reduce the parity game shown at the left below, which is minimal with respect to strong bisimulation and governed bisimulation, to the parity game shown to the right.


Stuttering equivalence, however, performs poorly in parity games with a high degree of alternation between vertices owned by different players. The parity game shown to the left in the following example is minimal using stuttering equivalence, however, it can easily be seen that player □ does not have any choice to make.

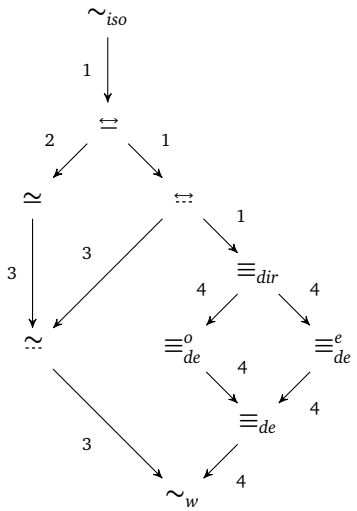



We therefore weaken the requirements of stuttering equivalence to allow for alternation between players. The relation that we obtain is dubbed *governed stuttering bisimilarity*, and relates both parity games in the picture above. Note that any equivalence notion, in the reduced game to the right above, may not relate both vertices with priority 0, since the rightmost vertex with priority 0 is won by player ◇, whereas the other is won by player □.

Governed stuttering bisimilarity, still requires related vertices to have the same priority. This modification is similar in spirit to the change from strong bisimulation to governed bisimulation. The main complication in relating vertices owned by different players is correctly treating divergences within equivalence classes containing vertices owned by different players.

We investigate the relationship between each of the equivalences that we have discussed. This gives rise to the lattice of equivalences depicted in Figure 4.1. For each of the edges in the lattice, we have indicated the theorem which shows the existence of the edge, and the corresponding strictness result. Note that this lattice includes winner equivalence ($\sim_w$), which is the coarsest meaningful equivalence on parity games, and isomorphism ($\sim_{iso}$), which is the finest meaningful equivalence on parity games.

The equivalences in this lattice have varying running time complexities. Deciding winner equivalence, which effectively means solving a parity game, is known to be in NP ∩ co-NP; most currently known algorithms require time exponential in the number of priorities in the game. The subexponential algorithm due to Jurdziński *et al.* [JPZ06] is a notable exception. In the following, assume that $n$, $m$, and $d$ are the numbers of vertices, edges, and priorities in a parity game, respectively. Deciding strong bisimilarity can be done in $\mathcal{O}(n \log m)$ time using Paige and Tarjan's partition refinement algorithm [PT87]. Existing strong bisimulation algorithms can straightforwardly be modified to decide governed bisimulation without resulting in a worse running time complexity. Fritz and Wilke showed that delayed simulation can be decided in $\mathcal{O}(n^3 d^2 m)$.




1. Theorem 4.19, page 82;
2. Theorem 4.35, page 89;
3. Theorem 4.45, page 93;
4. Theorem 4.8, page 80.

Figure 4.1: Lattice of equivalences for parity games. The numbers on the edges refer to the legend shown to the right, which in turn refers to the theorems that witness the existence of the edge.

Stuttering equivalence can be decided in $\mathcal{O}(nm)$ time using the partition refinement algorithm due to Groote and Vaandrager [GV90]. In addition to its complexity, stuttering equivalence has several other traits that make it appealing: quotienting is straightforward, distributed algorithms for computing stuttering equivalence have been developed (see *e.g.* [BO03]), and it admits efficient, scalable implementations using BDD technology [Wim+06].

In Section 4.5 we investigate an algorithm for deciding governed stuttering equivalence in $\mathcal{O}(n^2m)$ time, which is inspired by this algorithm. Note that modifying this algorithm to decide governed stuttering bisimilarity is not as straightforward as modifying the strong bisimilarity algorithm to decide governed bisimilarity due to the subtleties caused by divergence. In Section 4.5 we also indicate the case that is responsible for the extra factor $n$ in the worst-case running time complexity that we prove. The question whether governed stuttering equivalence can be decided in $\mathcal{O}(nm)$ time is still open.

**Structure of this chapter.** The remainder of this chapter is organised as follows. In Section 4.1 we first introduce some basic concepts that ease reasoning about parity games. We introduce winner equivalence and direct- and delayed simulation equivalence in Section 4.2. In Section 4.3 we recast strong bisimilarity and idempotence identifying bisimilarity in concepts natural to parity games. These simply specialise the equivalence presented in the previous chapter. Note that in the parity game setting we have dubbed the latter governed bisimulation, since idempotence is not a natural concept here. In Section 4.4 we define stuttering equivalence in the setting of parity games; we show that it can be used for minimising parity games. We weaken stuttering equivalence to allow

relating vertices owned by different players in Section 4.5, where we introduce the notion of governed stuttering equivalence.

## 4.1 Properties of Parity Games

Although it is convenient to reason about parity games in terms of players that move towards certain areas of the graph, it proves to be difficult to find a notation that defines intuitive concepts and also allows compact proofs. In this section, we introduce a notation that characterises the concept of a player being able to force the play towards a set of vertices. After that, some lemmata are given that express basic properties of parity games in terms of this extended notation.

We have several reasons to include these lemmata; first and foremost because they are instrumental for the proofs in this chapter, but also because we think that these lemmata are interesting in their own right, as they are fundamental and not always trivial to prove.

Throughout this whole section, fix some parity game $(V, \to, \Omega, \mathscr{P})$. Furthermore, we let $R \subseteq V \times V$ be a relation on vertices in the game.

We overload the notation used for the edge relation in a natural manner to deal with a set of vertices $U \subseteq V$:

$$v \to U \quad \triangleq \quad \exists u \in U : v \to u$$

The relation $R$ can be taken into account in the edge relation. This results in the edge relation $\to_R$ that only connects related vertices.

$$v \to_R u \quad \triangleq \quad v \to u \wedge v \, R \, u$$

We generalise the edge relation $\to_R$ to its transitive closure. Furthermore, we define what it means for a vertex to be divergent.

$$\begin{aligned} v \mapsto_R u \quad &\overset{\mu}{=} \quad v \to_R u \vee (\exists w \in V : v \to_R w \wedge w \mapsto_R u) \\ v \mapsto_R \quad &\overset{\nu}{=} \quad \exists u \in V : v \to_R u \wedge u \mapsto_R \end{aligned}$$

The fixed point notation above expresses that $\mapsto_R u$ and $\mapsto_R$ are least and greatest functions from (pairs of) vertices to Booleans, respectively, such that the equations hold. Intuitively, the first equation says that $v$ can reach $u$ through a finite number of $R$-related vertices, whereas the second states that from $v$ it is possible to take an infinite sequence of steps through $R$-related vertices.

This notation allows us to reason about *computation paths*. In the rest of this section we introduce notation that allows us to reason about *computation trees*.

Given a memoryless strategy $\sigma$ for some player, a move from vertex $v$ to another vertex $u$ may be allowed or disallowed by that strategy. We introduce the following notation:

$$v \, {}_\sigma\!\to u \quad \triangleq \quad \begin{cases} v \to u \wedge \sigma(v) = u, & \text{if } \sigma(v) \text{ is defined} \\ v \to u, & \text{otherwise} \end{cases}$$

We can now express that all plays allowed by $\sigma$ eventually reach some set $U$ from a vertex $v$, denoted $v \, {}_\sigma\!\mapsto U$. This notation is generalised to $v \, {}_\sigma\!\mapsto_R U$ to be able to express that additionally all plays allowed by $\sigma$ reach $U$ immediately when they follow an edge

to a vertex that is no longer related under relation $R$. The notation $v \,{}_\sigma\!\!\not\mapsto_R$ is in a sense the dual: it expresses that no play allowed by $\sigma$ can reach a vertex that is not related under $R$ to the previous vertex in that play:

$$\begin{aligned} v \,{}_\sigma\!\!\not\mapsto_R U &\stackrel{\mu}{=} \forall u : v \,{}_\sigma\!\!\to u \implies u \in U \vee (v \,R\, u \wedge u \,{}_\sigma\!\!\not\mapsto_R U) \\ v \,{}_\sigma\!\!\not\mapsto_R &\stackrel{\nu}{=} \forall u : v \,{}_\sigma\!\!\to u \implies v \,R\, u \wedge u \,{}_\sigma\!\!\not\mapsto_R \end{aligned}$$

In the same way as before, the fixed point notation above expresses that ${}_\sigma\!\!\not\mapsto_R U$ (resp. ${}_\sigma\!\!\not\mapsto_R$) is the least (resp. greatest) function from vertices to Booleans such that the equation holds, and corresponds essentially to the standard notions of inevitability and invariance. Using these definitions, we define what it means for a player to be able to force the play to a set of vertices $U$, or for a player to be able to force the play to *diverge* within a class of $R$:

$$\begin{aligned} x \,{}_i\!\!\not\mapsto_R U &\stackrel{\Delta}{=} \exists \sigma \in \mathbb{S}_i : x \,{}_\sigma\!\!\not\mapsto_R U \\ x \,{}_i\!\!\not\mapsto_R &\stackrel{\Delta}{=} \exists \sigma \in \mathbb{S}_i : x \,{}_\sigma\!\!\not\mapsto_R \end{aligned}$$

We leave out $R$ if $R$ is the relation that relates all vertices in $V$. Note that $v \,{}_i\!\!\not\mapsto_R \emptyset$ never holds, and that $v \,{}_i\!\!\not\mapsto_R V$ is trivially true. Furthermore, $v \,{}_i\!\!\not\mapsto$ is always true. We write $v \,{}_i\!\!\not\mapsto_R U$ for $\neg(v \,{}_i\!\!\mapsto_R U)$, and likewise for the other arrows. If $\mathscr{U} \subseteq V_{/R}$, then we write $v \,{}_i\!\!\mapsto_R \mathscr{U}$ to denote $v \,{}_i\!\!\mapsto_R \bigcup_{\mathscr{C} \in \mathscr{U}} \mathscr{C}$.

We are now ready to formalise some intuitions about parity games. One of the most basic properties we expect to hold is that a player can force the play towards some given set of vertices, or otherwise her opponent can force the play to the complement of that set.

**Lemma 4.1.** *Let $v \in V$, $U \subseteq V$, $i$ a player and $R$ an equivalence relation on $V$, then*

$$v \,{}_i\!\!\mapsto_R U \vee v \,{}_{\neg i}\!\!\mapsto_R V \setminus U.$$

*Proof.* We prove the equivalent $v \,{}_i\!\!\not\mapsto_R U \implies v \,{}_{\neg i}\!\!\mapsto_R V \setminus U$. Assume that $v \,{}_i\!\!\not\mapsto_R U$. We show that $v \,{}_{\neg i}\!\!\mapsto_R V \setminus U$. We distinguish on the player of $v$.

- $\mathscr{P}(v) = i$. As $v \,{}_i\!\!\not\mapsto_R U$, we know $\forall v' : v \to v' \implies v' \notin U$, hence also $\forall v' : v \to v' \implies v' \in V \setminus U$, so $v \,{}_{\neg i}\!\!\mapsto_R V \setminus U$.
- $\mathscr{P}(v) = \neg i$. As $v \,{}_i\!\!\not\mapsto_R U$, and the parity game is total, we know $\exists v' : v \to v' \wedge v' \notin U$. Let $v'$ be such, and define $\sigma \in \mathbb{S}_{\neg i}$ such that $\sigma(v) = v'$. $\sigma$ is a witness for $v \,{}_{\neg i}\!\!\mapsto_R V \setminus U$. □

In a similar train of thought, we expect that if from a single vertex, each player can force the play towards some target set, then the players' target sets must contain related vertices. Note that for this we need that $R$ is an equivalence relation. In particular, transitivity is used in the repetition of the argument.

**Lemma 4.2.** *Let $R$ be an equivalence relation, let $v \in V$, $U, U' \subseteq V$ and let $i$ be a player, then*

$$v \,{}_i\!\!\mapsto_R U \wedge v \,{}_{\neg i}\!\!\mapsto_R U' \implies \exists u \in U, u' \in U' : u \,R\, u' \vee u \,R\, v \vee u' \,R\, v.$$

*Proof.* Assume $v \,{}_i\!\mapsto_R U \wedge v \,{}_{\neg i}\!\mapsto_R U'$. Then there must be strategies $\sigma \in \mathbb{S}_i^*$ and $\sigma' \in \mathbb{S}_{\neg i}^*$ such that $v \,{}_\sigma\!\mapsto_R U \wedge v \,{}_{\sigma'}\!\mapsto_R U'$. Assume that $\mathscr{P}(v) = \neg i$ (the other case is symmetric). Then we have that $\sigma(v)$ is undefined and $\sigma'(v) = v'$ for some $v' \in V$. Obviously, $v \,{}_\sigma\!\to v'$ and $v \,{}_{\sigma'}\!\to v'$.

From the definitions of $v \,{}_\sigma\!\mapsto_R U$ and $v \,{}_{\sigma'}\!\mapsto_R U'$ we obtain $v' \in U \vee (v \mathrel{R} v' \wedge v' \,{}_\sigma\!\mapsto_R U)$ and $v' \in U' \vee (v \mathrel{R} v' \wedge v' \,{}_{\sigma'}\!\mapsto_R U')$. This leads to four cases:

1. $v' \in U \wedge v' \in U'$
2. $v' \in U \wedge (v \mathrel{R} v' \wedge v' \,{}_{\sigma'}\!\mapsto_R U')$
3. $(v \mathrel{R} v' \wedge v' \,{}_\sigma\!\mapsto_R U) \wedge v' \in U'$
4. $(v \mathrel{R} v' \wedge v' \,{}_\sigma\!\mapsto_R U) \wedge (v \mathrel{R} v' \wedge v' \,{}_{\sigma'}\!\mapsto_R U')$

The first three cases directly imply the desired result. The fourth case gives rise to a repetition of the same argument. The argument cannot be repeated infinitely long, because then $v \,{}_\sigma\!\mapsto_R U$ would not hold. □

If we consider sets of *classes* $\mathscr{U} \subseteq V_{/R}$ in $R$ that a player can force to, rather than arbitrary sets of vertices, we arrive at a much stronger version of Lemma 4.1; the result follows directly from Lemmata 4.1 and 4.2.

**Corollary 4.3.** $v \,{}_i\!\not\mapsto_R \mathscr{U} \iff v \,{}_{\neg i}\!\mapsto_R V_{/R} \setminus \mathscr{U}$.

The above lemmata reason about players being able to reach vertices. The following lemma is essentially about avoiding vertices: it states that if one player can force divergence, then this is the same as saying that the opponent cannot force the play outside the class of the current vertex.

**Lemma 4.4.** *Let $v \in V$, $i$ a player and $R$ an equivalence relation on $V$, then*

$$v \,{}_i\!\mapsto_R \iff v \,{}_{\neg i}\!\not\mapsto_R V \setminus [v]_R$$

*Proof.* Note that the truth values of $v \,{}_i\!\mapsto_R$ and $v \,{}_{\neg i}\!\not\mapsto_R V \setminus [v]_R$ only depend on edges that originate in $[v]_R$, and that these truth values do not depend on priorities at all. Therefore, the truth value of these predicates will not change if we apply the following transformations to our graph:

- For all $u \in V \setminus [v]_R$, replace all outgoing edges by a single edge $u \to u$.
- Make the priorities of all vertices in $[v]_R$ such that they are even iff $i = \Diamond$, and the priorities of all other vertices odd iff $i = \Diamond$.

In the resulting graph, player $i$ wins if and only if $v \,{}_i\!\mapsto_R$, and player $\neg i$ wins if and only if $v \,{}_{\neg i}\!\mapsto_R V \setminus [v]_R$. Since $v$ can only be won by one player, the desired result follows. □

Lastly, we want to formalise the idea that if a player can force the play to a first set of vertices, and from there he can force the play to a second set of vertices, then he must be able to force the play to that second set.

**Lemma 4.5.** *Let $v \in V$, $U, T \subseteq V$, $i$ a player and $R$ an equivalence relation.*

$$(v \,{}_i\!\!\mapsto_R U \wedge (\forall u \in U \setminus T : v \,R\, u \wedge u \,{}_i\!\!\mapsto_R T)) \implies v \,{}_i\!\!\mapsto_R T$$

*Proof.* Assume $v \,{}_i\!\!\mapsto_R U \wedge (\forall u \in U \setminus T : v \,R\, u \wedge {}_i\!\!\mapsto_R T)$. There must be a strategy $\sigma \in \mathbb{S}_i$ such that $v \,{}_\sigma\!\!\mapsto_R U$ and for each $u \in U \setminus T$ a strategy $\sigma_u \in \mathbb{S}_i$ such that $u \,{}_{\sigma_u}\!\!\mapsto_R T$. We define strategy $\sigma' \in \mathbb{S}_i^*$ as follows:

$$\sigma'(\pi v) = \begin{cases} \sigma(v) & \text{if } \forall u \in U \setminus T : u \notin \pi v \\ \sigma(v_u) & \text{if } \pi v = \pi' u \pi'' v \wedge u \in U \setminus T \wedge \forall u' \in U \setminus T : u' \notin \pi' \end{cases}$$

Observe that a memoryless strategy $\sigma'' \in \mathbb{S}_i$ can be found that has the same effect as $\sigma'$. Furthermore $v \,{}_{\sigma''}\!\!\mapsto_R T$, and hence $v \,{}_i\!\!\mapsto_R T$. □

Note that again the lemma is generalised to use a relation $R$. In practice, this $R$ can be used to provide extra information on the paths towards $U'$.

## 4.2 Winner Equivalence, Direct-, and Delayed Simulation

The coarsest meaningful equivalence on parity games is that of *winner equivalence*. We state that two vertices in a parity game are winner equivalent if and only if they are won by the same player. Essentially, every parity game reduced using winner equivalence has at most two equivalence classes, one containing vertices won by player $\diamond$, the other won by player $\Box$. This is effectively induced by determinacy of parity games.

**Definition 4.6.** Let $\mathcal{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. Two vertices $v, v' \in V$ are said to be *winner equivalent*, denoted $v \sim_w v'$ iff $v$ and $v'$ are won by the same player.

Because every vertex is won by exactly one player (see, *e.g.*, [Zie98]), winner equivalence partitions $V$ into a subset won by player $\diamond$ and a subset won by player $\Box$. Clearly, winner equivalence is an equivalence relation on the set of vertices of a given parity game.

Direct simulation preorder and its weaker versions are defined along the lines of Stirling's bisimulation games [Sti97], through a game played on an auxiliary graph referred to as simulation game-graph. We here use the exposition of direct simulation preorder as given by Gazda and Willemse [GW12]. The simulation game for a parity game $(V, \to, \Omega, \mathscr{P})$ is played by the players *Duplicator* and *Spoiler*, further referred to as $D$ and $S$, respectively. The game graph $(V', \to')$ has vertices in $V \times V$ and edges in $\to \times \to$. There is an edge from $(v, w) \to' (v', w')$ if $v \to v'$ and $w \to w'$. The moves are made according to the rules in Table 4.1.

*Duplicator* wins an infinite play $(v_0, w_0), (v_1, w_1), \ldots$ if, for all $k$, $\Omega(v_k) = \Omega(w_k)$, *i.e.*, *Duplicator* was always able to mimic the moves with a move to a vertex with equal priority. In all other cases the play is won by *Spoiler*. Formally, direct simulation preorder is defined as follows.

**Definition 4.7** ([FW06; GW12])**.** Let $(V, \to, \Omega, \mathscr{P})$ be a parity game. Vertex $v$ is directly simulated by $w$, denoted $v \sqsubseteq_{dir} w$, if *Duplicator* has a winning strategy for the simulation game starting in $(v, w)$. We say that $v$ and $w$ are direct simulation equivalent, denoted $v \equiv_{dir} w$, if $v \sqsubseteq_{dir} w$ and $w \sqsubseteq_{dir} v$.

| $(v,w) \in$ | $1^{st}$ move | plays on | $2^{nd}$ move | plays on |
|---|---|---|---|---|
| $V_\Diamond \times V_\Diamond$ | S | $v$ | D | $w$ |
| $V_\Diamond \times V_\Box$ | S | $v$ | S | $w$ |
| $V_\Box \times V_\Diamond$ | D | $w$ | D | $v$ |
| $V_\Box \times V_\Box$ | S | $w$ | D | $v$ |

Table 4.1: Allowed moves in a simulation game, where $V_i = \{v \in V \mid \mathscr{P}(v) = i\}$.

The notion of delayed simulation, denoted $\sqsubseteq_{de}$ with its corresponding equivalence $\equiv_{de}$, introduced by Fritz and Wilke, weakens direct simulation. For delayed simulation a modified arena is used for playing the game, in which they keep track of the obligations that still need to be fulfilled, *i.e.*, the steps that still need to be mimicked, but that were postponed so far. They also introduce *even*- and *odd* biased versions of delayed simulation, that we denote $\sqsubseteq^e_{de}$ and $\sqsubseteq^o_{de}$, with equivalences $\equiv^e_{de}$ and $\equiv^o_{de}$, respectively. We refrain from presenting the details of delayed simulation, but we just state the main results from [FW06] that are of interest to this section.

**Theorem 4.8** ([FW06])**.** *We have the following relationships between the equivalences presented in [FW06].*

- $\equiv_{de}$ *is strictly finer than* $\sim_w$*;*
- $\equiv^e_{de}$ *and* $\equiv^o_{de}$ *are strictly finer than* $\equiv_{de}$*;*
- $\equiv_{dir}$ *is strictly finer than* $\equiv^e_{de}$ *and* $\equiv^o_{de}$*.*

## 4.3 Strong Bisimilarity and Governed Bisimilarity

In the previous chapter we have introduced the notions of solution equivalence, strong bisimilarity and idempotence identifying bisimilarity. In this section we rephrase them in the terminology native to parity games. We basically interpret the priorities and players of vertices as state labels, and we define strong bisimilarity analogously to strong bisimilarity in the previous chapter.

**Definition 4.9** (Strong bisimulation)**.** Let $\mathscr{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. A symmetric relation $R \subseteq V \times V$ is a *strong bisimulation* relation if $v \mathrel{R} v'$ implies

- $\Omega(v) = \Omega(v')$ and $\mathscr{P}(v) = \mathscr{P}(v')$;
- for all $w \in V$ such that $v \to w$, there should be a $w' \in V$ such that $v' \to w'$ and $w \mathrel{R} w'$.

Vertices $v$ and $v'$ are said to be *strongly bisimilar*, denoted $v \leftrightarrows v'$, if and only if a strong bisimulation relation $R$ exists such that $v \mathrel{R} v'$.

The following results, showing that strong bisimilarity is an equivalence relation on vertices; that quotienting with respect to strong bisimilarity is well-defined, and that it is strictly finer than winner equivalence, follow immediately from the results in the previous chapter.

**Proposition 4.10.** *Relation $\leftrightarrows$ is an equivalence relation on parity games.*

This specialises Proposition 3.3. The definition of quotienting is more compact than it is for structure graph, because we do not have to consider free variables and Boolean constants.

**Definition 4.11.** Let $(V, \to \Omega, \mathscr{P})$ be a parity game. Its strong bisimulation quotient is the parity game $(V_m, \to_m, \Omega_m, \mathscr{P}_m)$ adhering to the following:

- $V_m = \{[v]_{\leftrightarrows} \mid v \in V\}$,
- $\Omega_m([v]_{\leftrightarrows}) = \Omega(v)$,
- $\mathscr{P}_m([v]_{\leftrightarrows}) = \mathscr{P}(v)$, and
- $[v]_{\leftrightarrows} \to [v']_{\leftrightarrows}$ iff $v \to v'$.

Using the standard algorithm for deciding strong bisimulation due to Paige and Tarjan, quotienting with respect to strong bisimulation can be done effectively.

**Theorem 4.12** ([PT87])**.** *Relation $\leftrightarrows$ can be decided in $\mathcal{O}(n \log n)$, where $n$ is the number of vertices in the game.*

In the previous chapter we showed how, in structure graphs, vertices labelled ▲ and ▼ can sometimes be related using the notion of idempotence identifying bisimilarity. We can directly carry this notion over to parity games. In this setting, instead of looking at idempotence of the formulae in a label, we look at the choices that a player can make in a vertex. If all successors of a vertex are related, the choice for a successor vertex in the quotient is effectively forced. Therefore, two vertices may be related if *all* of their successors are related. We dub the relation that we thus obtain *governed bisimulation*.

**Definition 4.13** (Governed bisimulation)**.** Let $\mathscr{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. A symmetric relation $R \subseteq V \times V$ is a *governed bisimulation* relation if $v \mathrel{R} v'$ implies

- $\Omega(v) = \Omega(v')$;
- $\mathscr{P}(v) \neq \mathscr{P}(v')$ implies that for all $w, w' \in V$ such that $v \to w$ and $v' \to w'$, it is the case that $w \mathrel{R} w'$; and
- for all $w \in V$ such that $v \to w$, there should be a $w' \in V$ such that $v' \to w'$ and $w \mathrel{R} w'$.

Vertices $v$ and $v'$ are said to be *governed bisimilar*, denoted $v \rightleftarrows v'$, if and only if a governed bisimulation relation $R$ exists such that $v \mathrel{R} v'$.

Note that governed bisimulation is an equivalence relation on parity games. This follows immediately from Proposition 3.42.

**Theorem 4.14.** *Relation $\rightleftarrows$ is an equivalence relation on parity games.*

Quotienting using governed bisimilarity is again well-defined. However, we need a suitable ordering on players to choose a representative vertex in case vertices owned by different players are in the same equivalence class. Let $\lessdot$ be this ordering on players such that $\Box \lessdot \Diamond$, where $\min_{\lessdot}$ is the corresponding minimum.

**Definition 4.15.** Let $(V, \to \Omega, \mathscr{P})$ be a parity game. Its governed bisimulation quotient is the parity game $(V_m, \to_m, \Omega_m, \mathscr{P}_m)$ adhering to the following:

- $V_m = \{[v]_{\leftrightarrows} \mid v \in V\}$,
- $\Omega_m([v]_{\leftrightarrows}) = \Omega(v)$,
- $\mathscr{P}_m([v]_{\leftrightarrows}) = \min_{\lessdot}\{\mathscr{P}(v') \mid v' \in [v]_{\leftrightarrows}\}$, and
- $[v]_{\leftrightarrows} \to [v']_{\leftrightarrows}$ iff $v \to v'$.

Algorithms that decide strong bisimilarity can easily be modified to yield an algorithm for deciding governed bisimilarity without a penalty in terms of complexity.

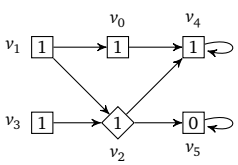
**Theorem 4.16.** *Relation $\leftrightarrows$ can be decided in $\mathcal{O}(n \log n)$ time, where $n$ is the number of vertices in the parity game.*

We next investigate how strong bisimilarity and governed bisimilarity compare to direct- and delayed simulation equivalence. Gazda and Willemse [GW12] introduced consistent consequences, which are simulation relations for parameterised Boolean equation systems. They showed that, for the specialised case of parity games, consistent consequences coincide with direct simulation. From the results in [GW12], we immediately obtain the following correspondence.

**Proposition 4.17** ([GW12, Proposition 2])**.** *Let $\mathcal{G}$ be an arbitrary parity game. The largest governed bisimulation on $\mathcal{G}$ is the largest symmetric direct simulation on $\mathcal{G}$, and is contained in $\equiv_{dir}$. Furthermore, the largest governed bisimulation on $\mathcal{G}$ is strictly finer than $\equiv_{dir}$.*

The following example was used by Gazda an Willemse to show strictness in the above proposition.

**Example 4.18.** Consider the following parity game.




In this game, among others $v_0 \sqsubseteq_{dir} v_2$, $v_1 \sqsubseteq_{dir} v_3$, $v_3 \sqsubseteq_{dir} v_1$, and $v_4 \sqsubseteq_{dir} v_0$. However, $v_1 \not\leftrightarrows v_3$, hence governed bisimulation is strictly finer than the symmetric version of direct simulation.

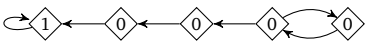
We derive the following correspondence between the equivalences directly from Theorem 3.51 and Proposition 4.17.

**Theorem 4.19.** *We have the following relationship between the equivalences on parity games from this section:*

- $\sim_{iso}$ *is strictly finer than* $\leftrightarrows$,
- $\leftrightarrows$ *is strictly finer than* $\leftrightarrows$,

- $\leftrightarrows$ *is strictly finer than* $\equiv_{dir}$, *and*
- $\leftrightarrows$ *is strictly finer than* $\sim_w$.

The major drawback of both strong bisimilarity and governed bisimilarity is their sensitivity to counting—in the sense that they will not identify vertices that require a different number of steps to reach a next equivalence class—preventing them from compressing the game graph any further. Recall, *e.g.*, the following parity game. This game is minimal with respect to strong bisimulation and governed bisimulation; it is, however, not hard to verify that the rightmost vertices with priority 0 are won by $\Diamond$, and the other two vertices with priority 0 are won by player $\Box$. We might therefore want to relate these vertices.




In the following two sections we introduce weaker equivalences that indeed allow us to reduce such sequences of vertices.

## 4.4 Stuttering Equivalence

Stuttering equivalence shares many of the characteristics of strong bisimilarity, and deciding it has only a slightly worse worst-case time complexity. However, it is insensitive to counting, and is therefore likely to lead to greater reductions. Given these observations, we hypothesise that using stuttering equivalence parity games can often be reduced significantly further than using strong bisimilarity.

We first introduce stuttering bisimilarity [NV95], a coinductive alternative to the stuttering equivalence of Browne, Clarke and Grumberg; we shall use the terms stuttering bisimilarity and stuttering equivalence interchangeably. We restate the well-known result that stuttering bisimilarity is coarser than strong bisimilarity. The remainder of this section is then devoted to showing that it is still finer than winner equivalence. The latter result allows one to pre-process a parity game by quotienting it using stuttering equivalence.

Stuttering bisimulation is defined using the notation on computation paths introduced in the previous section.

**Definition 4.20** (Stuttering bisimulation)**.** Let $\mathcal{G} = (V, \to, \Omega, \mathscr{P})$ be a parity game. Let $R \subseteq V \times V$ be a symmetric relation on vertices; $R$ is a *stuttering bisimulation* if $v \; R \; v'$ implies

- $\Omega(v) = \Omega(v')$ and $\mathscr{P}(v) = \mathscr{P}(v')$;
- if $v \to u$ then either:
  - $(v \, R \, u \wedge u \, R \, v')$, or
  - there are $u', w$ such that $v' \mapsto_R w \to u'$ and $v \, R \, w \wedge u \, R \, u'$;
- $v \mapsto_R$ iff $v' \mapsto_R$.

Two states $v$ and $v'$ are said to be *stuttering bisimilar*, denoted $v \simeq v'$ iff there is a stuttering bisimulation relation $R$, such that $v \, R \, v'$.

Stuttering bisimilarity adheres to the following key property, that was proven for Kripke structures, and carries over to the setting of parity games.

**Theorem 4.21** ([NV95; BCG88])**.** *Relation $\simeq$ is an equivalence relation.*

Proving transitivity of stuttering bisimilarity is particularly difficult, but the result is well-known. Quotienting with respect to stuttering bisimilarity is formally defined as follows.

**Definition 4.22.** The stuttering bisimulation quotient of a parity game $(V, \to, \Omega, \mathscr{P})$ is defined as a game $(V', \to', \Omega', \mathscr{P}')$, that satisfies the following conditions:

- $V' = \{[v]_\simeq \mid v \in V\}$
- $\Omega'([v]_\simeq) = \Omega(v)$
- $\mathscr{P}'([v]_\simeq) = \mathscr{P}(v)$
- $[v]_\simeq \to' [v]_\simeq$ iff $v \mapsto_\simeq$
- $[v]_\simeq \to' [v']_\simeq$ iff there exists a $w$ such that $v \mapsto_\simeq w \to v'$ and $[v']_\simeq \neq [v]_\simeq$

Note that, in this definition, the priorities and players of the quotient vertices are uniquely defined due to the first requirement of stuttering bisimulation, *i.e.*, related vertices always have the same priority and the same player.

Decidability of stuttering equivalence is well-known.

**Theorem 4.23** ([GV90])**.** *$\simeq$ can be decided in $\mathcal{O}(nm)$ time, where $n$ is the number of vertices, and $m$ the number of edges in the game.*

Stuttering bisimilarity between vertices extends naturally to finite paths. Paths of length 1 are equivalent if the vertices they consist of are equivalent. If paths $p$ and $q$ are equivalent, then $p \cdot \langle v \rangle \simeq q$ iff $v$ is equivalent to the last vertex in $q$. Equivalence of extensions $q \cdot \langle w \rangle$ with $p$ is defined analogously. Finally, if both paths are extended, we have $p \cdot \langle v \rangle \simeq q \cdot \langle w \rangle$ iff $v \simeq w$. An infinite path $p$ is equivalent to a (possibly infinite) path $q$ if for all finite prefixes of $p$ there is an equivalent prefix of $q$ and *vice versa*. Recall that $\Pi^n_\sigma(v)$ is the set of paths of length $n$ starting in vertex $v$, allowed by strategy $\sigma$, and $\Pi^\omega_\sigma(v)$ denotes the infinite paths starting in $v$, consistent with $\sigma$.

We aim to prove that stuttering equivalence refines winner equivalence. Our proof relies on the observation that, for vertices $v, w$ that are related using stuttering equivalence, strategy $\sigma$ and an infinite path starting in $v$, consistent with $\sigma$, we can construct a mimicking strategy $\psi$ for which there is a path starting in $w$, that is consistent with $\psi$ such that both paths are stuttering equivalent. This is formalised by the following proposition.

**Proposition 4.24.** *Given some $v, w \in V$ such that $v \simeq w$, then for every strategy $\sigma \in \mathbb{S}^*_i$ we have a strategy $\psi \in \mathbb{S}^*_i$ such that*

$$\forall p \in \Pi^\omega_\psi(w) : \exists q \in \Pi^\omega_\sigma(v) : p \simeq q.$$

This result immediately proves that stuttering bisimilarity is finer than winner equivalence: if a vertex $v$ is won by player $\Diamond$, a winning strategy $\sigma$ for player $\Diamond$ from that vertex exists; for all related vertices $w$ we can construct a mimicking strategy $\psi$, such that all infinite paths consistent with $\psi$ are related to an infinite path consistent with $\sigma$; this means that $\psi$ is also winning for player $\Diamond$. The symmetric argument holds for vertices won by player $\Box$.

Before we go on to prove this proposition, we first show how to explicitly construct such a mimicking strategy, and we study its properties. Afterwards we return to the proof of this proposition.

In the following, assume that there is a strategy $\sigma$ for player $i$ from a vertex $v$. We define a strategy $\mathsf{mimic}_{\sigma,v}$, that from vertices stuttering equivalent to $v$ schedules only paths that are stuttering bisimilar to a path starting in $v$ that is consistent with $\sigma$.

Let $\sigma$ be a strategy, $v$ a vertex owned by the player for which $\sigma$ defines a strategy, and let $p$ be an arbitrary path. The set $\mathsf{reach}_{\sigma,v}(p)$ determines the set of vertices $u$ in new classes reachable by traversing $\sigma$ consistent paths that start in $v$ and that are stuttering bisimilar to $p$. Note that if this set is empty, then there is no stuttering equivalent path $q$ that can be extended to a vertex $u$ that is not equivalent to the last vertex of $p$. This means that $p$ must be divergent.

**Definition 4.25.** Let $i$ be a player, with $\sigma \in \mathbb{S}_i^*$, let $v \in V$ such that $\mathscr{P}(v) = i$, and let $p$ be an arbitrary path consistent with $\sigma$.

$$\mathsf{reach}_{\sigma,v}(p) = \{u \in V \mid \exists m \in \mathbb{N} : \exists q \in \Pi_\sigma^m(v) : p \simeq q \wedge \sigma \Vdash q \cdot \langle u \rangle \wedge q \cdot \langle u \rangle \not\simeq q\}$$

Observe that not all vertices in $\mathsf{reach}_{\sigma,v}(p)$ have to be in the same equivalence class, because it is not guaranteed that all paths $q \in \Pi_\sigma(v)$, stuttering bisimilar to $p$, are extended by $\sigma$ towards the same equivalence class.

In the following two definitions suppose that $\mathsf{reach}_{\sigma,v}(p)$ is non-empty; the case where it is empty is treated separately in Definition 4.28.

The strategy that we construct should select a *target class* to which $p$ should be extended. Because stuttering bisimilar vertices can reach the same classes, it does not matter which class present in $\mathsf{reach}_{\sigma,v}(p)$ is selected as the target class. We do however need to make a unique choice to guarantee that all stuttering bisimilar paths are extended to the same path by our strategy; to this end we use the ordering $\sqsubset$ on vertices, and its minimum $\sqcap$, introduced in Section 2.6.

**Definition 4.26.** Let $i$ be a player with strategy $\sigma \in \mathbb{S}_i^*$, let $v \in V$ such that $\mathscr{P}(v) = i$, and let $p$ be an arbitrary path consistent with $\sigma$, then

$$\mathsf{targetclass}_{\sigma,v}(p) = [\sqcap(\mathsf{reach}_{\sigma,v}(p)]_\simeq$$

is the target class to which $p$ should be extended.

Not all vertices in the target class have to be reachable from $p$, but there exists at least one vertex that is reachable due to the definition of $\mathsf{reach}_{\sigma,v}(p)$, and the stuttering equivalence it uses. We next determine a *target vertex*, by selecting a unique, reachable vertex in the target class. This target of $p$, given a strategy $\sigma$ and a vertex $v$ is denoted $\tau_{\sigma,v}(p)$.

**Definition 4.27.** Let $i$ be a player with strategy $\sigma \in \mathbb{S}_i^*$, let $v \in V$ such that $\mathscr{P}(v) = i$, and let $p$ be an arbitrary path consistent with $\sigma$, then $\tau_{\sigma,v}(p)$ is the vertex satisfying all of the following:

1. $\tau_{\sigma,v}(p) \in \mathsf{targetclass}_{\sigma,v}(p)$,
2. $\forall u \in \mathsf{targetclass}_{\sigma,v}(p) : (\exists w \in V : p \mapsto_{\simeq} w \to u) \implies \tau_{\sigma,v}(p) \sqsubset u$, and
3. $\exists w \in V : p \mapsto_{\simeq} w \to \tau_{\sigma,v}(p)$.

Note that in the above definition the ordering $\sqsubset$ is again used to uniquely determine a vertex from the set of reachable vertices. The third clause ensures that $\tau_{\sigma,v}(p)$ is reachable from $p$, and, in fact, that it is an element of $\mathsf{reach}_{\sigma,v}(p)$.

Finally, we are ready to define a strategy $\mathsf{mimic}_{\sigma,v}$ for player $i$ that, given some strategy $\sigma$ for player $i$ and a vertex $v$ allows only paths to be scheduled that have a stuttering bisimilar path starting in $v$ that is scheduled by $\sigma$.

Let $|v,u|$ denote the least number of edges required to move from vertex $v$ to vertex $u$ in the graph. We define $|v,u| = \infty$ if $u$ is unreachable from $v$. For each vertex $u \in V$, we define an ordering $\prec_u \subseteq V \times V$ on vertices, that intuitively orders vertices based on their proximity to $u$, with a subjugate role for the vertex ordering $\sqsubset$:

$$v \prec_u v' \text{iff} |v,u| < \left|v',u\right| \text{ or } (|v,u| = \left|v',u\right| \text{ and } v \sqsubset v')$$

Observe that $u \prec_u v$ for all $v \neq u$. The minimal element of $U \subseteq V$ with respect to $\prec_u$ is written $\lambda_u(U)$.

**Definition 4.28.** Let $i$ be a player with strategy $\sigma \in \mathbb{S}_i^*$, let $v \in V$ such that $\mathscr{P}(v) = i$, and let $p$ be an arbitrary path consistent with $\sigma$, then mimic is defined as follows.

$$\mathsf{mimic}_{\sigma,v}(p) = \begin{cases} \lambda_t\{u \in V \mid p \to_{\simeq} u\}, & t = \tau_{\sigma,v}(p) \\ & p \not\to \tau_{\sigma,v}(p) \\ & \mathsf{reach}_{\sigma,v}(p) \neq \emptyset \\ \tau_{\sigma,v}(p) & p \to \tau_{\sigma,v}(p) \\ & \mathsf{reach}_{\sigma,v}(p) \neq \emptyset \\ \sqcap\{u \in V \mid p \to_{\simeq} u\}, & \mathsf{reach}_{\sigma,v}(p) = \emptyset \end{cases}$$

The mimicking strategy constructed in this definition is, intuitively, built as follows. If $\mathsf{reach}_{\sigma,v}(p) = \emptyset$ this means that $p$ diverges; then the mimicking strategy is constructed such that it selects the least successor of $p$ that is stuttering equivalent. If $\mathsf{reach}_{\sigma,v}(p) \neq \emptyset$ we consider two cases. If $p$ can be extended to the target of $p$ given $\sigma$ and $v$ in one step, then we select this target. In case the target is not reachable from $p$ in a single step, then we choose the stuttering equivalent successor of $p$ that is the closest to this target. Using the ordering on targets in this definition effectively ensures that progress is made towards the target. In the following lemma we prove that the mimicking strategy is able to mimic finite paths $p$ consistent with strategy $\sigma$ starting in $v$.

**Lemma 4.29.** *Let $i$ be a player, with $\sigma \in \mathbb{S}_i^*$ an arbitrary strategy in a parity game. Assume that $v, w \in V$ and $v \simeq w$, and let $\psi = \mathsf{mimic}_{\sigma,v}(p)$. Then*

$$\forall l \in \mathbb{N} : \forall p \in \Pi_{\psi}^{l+1}(w) : \ \exists k \in \mathbb{N} : \exists q \in \Pi_{\sigma}^{k}(v) : \ p \simeq q$$

*Proof.* We proceed by induction on $l$. For $l = 0$, the desired implication follows immediately. For $l = n+1$, assume that we have a path $p \in \Pi^{n+2}_{\psi}(w)$. Clearly, $\langle p_1, \ldots, p_n \rangle$ is also consistent with $\psi$. The induction hypothesis yields us, for some $k \in \mathbb{N}$, a $q \in \Pi^{k}_{\sigma}(v)$ such that $\langle p_1, \ldots, p_n \rangle \simeq q$. Let $q$ be such. We distinguish the following cases:

1. $p_n \simeq p_{n+1}$. In this case, clearly $p \simeq \langle p_1, \ldots, p_n \rangle \simeq q$, which finishes this case.
2. $p_n \not\simeq p_{n+1}$. We again distinguish two cases:
   (a) $\mathscr{P}(p_n) \neq i$. Since $p_n \simeq q_k$, we find that there must be states $u, w \in V$ such that $q_k \mapsto_{\simeq} w \to u$ and $p_{n+1} \simeq u$. So there must be a path $r$ and vertex $u$ such that $p \simeq q \cdot r \cdot \langle u \rangle$, for which we know that $r \simeq q_k$. Therefore, all vertices in $r$ are owned by $\mathscr{P}(q_k) = \mathscr{P}(p_n)$, so $\sigma$ is not defined for the extensions of $q$ along $p$. We can therefore conclude that $\sigma \Vdash q \cdot r \cdot \langle u \rangle$.
   (b) $\mathscr{P}(p_n) = i$. Then it must be the case that $p_{n+1} = \tau_{\sigma,v}(\langle p_1, \ldots, p_n \rangle)$. By definition, that means that there is a $\sigma$-consistent path $r \in \Pi_{\sigma}(v)$, such that $r \simeq p$.

□

It is now time to revisit Proposition 4.30. We restate the proposition, and prove that that $\mathsf{mimic}_{\sigma,v}$ is a witness in this proposition.

**Proposition 4.30.** *Given some $v, w \in V$ such that $v \simeq w$, then for every strategy $\sigma \in \mathbb{S}^{*}_{i}$ we have a strategy $\psi \in \mathbb{S}^{*}_{i}$ such that*

$$\forall p \in \Pi^{\omega}_{\psi}(w) : \exists q \in \Pi^{\omega}_{\sigma}(v) : p \simeq q.$$

*Proof.* Let $v, w \in V$ be arbitrary such that $v \simeq w$, and consider an arbitrary strategy $\sigma$. Let $\psi = \mathsf{mimic}_{\sigma,v}$. We show that $\psi$ is a witness of the existence of a strategy such that

$$\forall p \in \Pi^{\omega}_{\psi}(w) : \exists q \in \Pi^{\omega}_{\sigma}(v) : p \simeq q.$$

Suppose we have an infinite path $p \in \Pi^{\omega}_{\psi}(w)$. Using Lemma 4.29 we can obtain a path $q$ starting in $v$ that is stuttering bisimilar, and that is consistent with $\sigma$. The lemma does not guarantee, however, that $q$ is of infinite length. We show that if $q$ is finite, it can always be extended to an infinite path that is still consistent with $\sigma$.

Notice that paths can be partitioned into segments of vertices from the same equivalence class, and that two stuttering bisimilar paths must have the same number of segments. This also follows from the original definition of stuttering equivalence given in [BCG88].

Suppose now that $q$ is of finite length, say $k+1$. Then $p$ must contain such a segment that has infinite size. In particular, there must be some $n \in \mathbb{N}$ such that $p_{n+j} \simeq p_{n+j+1}$ for all $0 \leq j \leq |V|$. We distinguish two cases.

1. $\mathscr{P}(p_n) = i$. We show that then also $\mathsf{reach}_{\sigma,v}(\langle p_0, p_1, \ldots p_n \rangle) = \emptyset$. Suppose this is not the case. Then we find that for some $u \in V$, $u = \tau_{\sigma,v}(p)$ exists, and therefore $p_{n+j} \prec_u p_{n+j+1}$ for all $j \leq |V|$. Since $\prec_u$ is total, this means that the longest chain is of length $|V|$, which contradicts our assumptions. So, necessarily $\mathsf{reach}_{\sigma,v}(\langle p_0, p_1, \ldots p_n \rangle) = \emptyset$, meaning that no path that is consistent with $\sigma$ leaves the class of $p_n$. But this means that the infinite path that stays in the class of $p_n$ is also consistent with $\sigma$.

2. $\mathscr{P}(p_n) \neq i$. Since $p_n \simeq q_k$, also $\mathscr{P}(q_k) \neq i$. Since $p_n \simeq p_{n+j}$ for all $j \leq |V|+1$, this means that there is a state $u$, and $l, l'$, such that $u = p_{n+l} = p_{n+l'}$. But this means that $u$ is divergent. Since $\mathscr{P}(u) \neq i$, and $u \simeq q_k$, we find that also $q_k$ is divergent. Therefore, there is an infinite path with prefix $q$ that is consistent with $\sigma$ and that is stuttering bisimilar to $p$. □

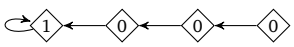
As we described before, this effectively proves that stuttering equivalence refines winner equivalence.

**Theorem 4.31.** *Let $v, w \in V$, then $v \simeq w$ implies $v \sim_w w$.*

*Proof.* Let $v, w \in V$ be vertices such that $v \simeq w$. Without loss of generality, assume that $v$ is won by $\diamond$. There exists a winning strategy $\sigma$ for player $\diamond$ from $v$ vertex; let $p$ be an arbitrary infinite path, consistent with $\sigma$. According to Proposition 4.30, there exists an infinite path $q$, consistent with $\psi = \mathsf{mimic}_{\sigma,v}$ such that $p \simeq q$. Since stuttering equivalent infinite paths have the same priorities that occur infinitely often, $p$ and $q$ are won by the same player. The symmetric argument holds for vertices won by player □. □

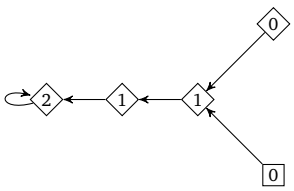
Strictness is shown in the following example.

**Example 4.32.** Consider the parity game below. All vertices in this game are winner equivalent —they are won by player □—but the vertex with priority 1 is not related to the vertices with priority 0 under stuttering equivalence.




Stuttering equivalence and governed bisimulation are incomparable. This is demonstrated in the following example.

**Example 4.33.** Consider the parity game below. All vertices with priority 1 can be related using stuttering equivalence, and not using governed bisimulation, on the other hand, the vertices with priority 0 can be related using governed bisimulation, but not using stuttering equivalence.




The parity game in Example 4.18, that we used to show that direct simulation is finer than governed bisimulation also illustrates that direct simulation is able to relate vertices that cannot be related using stuttering equivalence, since $v_1 \not\simeq v_3$ in that example. On the other hand, stuttering equivalence can relate vertices that cannot be related using delayed simulation, as is illustrated by the following example.

**Example 4.34.** Consider the parity games below. In both parity games, all vertices with equal priority can be related using stuttering equivalence, but they cannot be related by direct simulation equivalence.


We summarise the relationships of stuttering bisimilarity with the other equivalences in the following theorem.

**Theorem 4.35.** *For stuttering bisimilarity we have the following results:*

- *Strong bisimilarity strictly refines stuttering bisimilarity, i.e.,* $\leftrightarrows \subseteq \simeq$*;*
- *stuttering bisimilarity strictly refines winner equivalence, i.e.,* $\simeq \subseteq \sim_w$*;*
- *stuttering bisimilarity and governed bisimilarity are incomparable;*
- *stuttering bisimilarity is incomparable with direct simulation, delayed simulation, and the biased variations of delayed simulation.*

*Proof.* The first result is well-known, and follows immediately from the observation that in stuttering equivalence we just weaken the third condition of strong bisimulation. Strictness follows from the parity game in Example 4.32, which is minimal modulo strong bisimilarity, but in which all vertices with priority 0 can be related using stuttering bisimilarity.

The observation that stuttering bisimilarity is finer than winner equivalence follows immediately from Theorem 4.31. Strictness follows from the game in Example 4.32.

Incomparability of stuttering bisimilarity and governed bisimilarity follows from the game in Example 4.33. Finally, incomparability of stuttering bisimilarity with the equivalences by Fritz and Wilke follows from Examples 4.18 and 4.34. □

Note that the proof that strong bisimilarity strictly refines winner equivalence can now be simplified using transitivity of the above.

As an aside, we point out that our proof of Proposition 4.30 relies on the construction of the strategy mimic; its purpose, however, exceeds that of the proof. If, by solving the stuttering bisimilar quotient of a given parity game $\mathcal{G}$, one obtains a winning strategy $\sigma$ for a given player, mimic defines the winning strategies for that player in $\mathcal{G}$. This is of particular importance in case an explanation of the solution of the game is required, for instance when the game encodes a verification problem for which a strategy helps explain the outcome of the verification (see *e.g.* [SS98]). It is not immediately obvious how a similar feature could be obtained in the setting of, say, the delayed simulations of Fritz and Wilke [FW06], because vertices that belong to different players and that have different priorities can be identified through such simulations.

## 4.5 Governed stuttering equivalence

We have adapted the notion of stuttering bisimilarity to the setting of parity games, and proven that this equivalence relation can be safely used to minimise a parity game before solving the reduced game.

Stuttering bisimulation, however, is inept when faced with alternations between players along the possible plays: it cannot relate vertices belonging to different players. Turn-based games, controller synthesis problems, see *e.g.* [AVW03], tree automata emptiness,

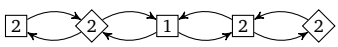
and, in general, constructs such as $\Box\Diamond\phi$ and $\Diamond\Box\phi$ in $\mu$-calculus verification, all give rise to such parity games.

Strong bisimulation was modified to governed bisimulation to relate vertices owned by different players. The resulting equivalence is incomparable to stuttering equivalence. A natural question is, therefore, whether stuttering bisimulation can also be modified so that it is able to relate vertices that belong to different players, resulting in a relation that subsumes both governed bisimulation as well as stuttering equivalence. Here we report on the research conducted in [CKW12b], showing that, indeed, it is possible to relax this definition. We give the definition of the resulting relation *governed stuttering bisimulation*, along with its key properties. In this chapter, however, we focus on the algorithmic aspects of deciding governed stuttering bisimilarity.[1]

We show that governed stuttering bisimilarity is decidable in $\mathcal{O}(n^2m)$ time using a partition refinement algorithm that specialises Groote and Vaandrager's algorithm for deciding stuttering equivalence. Observe that the time complexity for deciding governed stuttering bisimilarity is a factor $n$ worse than that for stuttering bisimilarity; this is due to a single type of class in a partition for which our algorithm requires $\mathcal{O}(mn)$ rather than $\mathcal{O}(m)$ time to check its stability. We hypothesise, however, that in most practical cases this factor does not manifest itself. This hypothesis is further investigated in the next chapter, where we investigate both the effect of all equivalences that we defined on the sizes of the parity games, as well as the practical impact they have on solving parity games.

In [CKW12b], the objective was to weaken stuttering bisimulation so that it is able to relate vertices of different players. However, simply weakening clause *a)* in Definition 4.20 to $\Omega(v) = \Omega(v')$ without modifying the remaining clauses, does not suffice as this would enable us to relate vertices won by different players. This is illustrated by the following example.

**Example 4.36.** Consider the following parity game.




The suggested weakening would allow us to relate all vertices with priority 2; the two left vertices, however are won by player $\Diamond$, whereas the other vertices are won by player $\Box$.

The problem in the above example is that the computation paths that appear in clauses *b)* and *c)* may consist of vertices owned by different players. This means that a fixed player is at the mercy of her opponent to stay on a computation path: the opponent may simply choose an alternative next vertex if that would better suit her. We are therefore forced to reason about computation trees, taking all of the opponent's choices into account. Effectively, clause *b)* must be strengthened to ensure that a player eventually reaches class $\mathscr{C}$ along some computation tree, and clause *c)* must be strengthened to ensure that a player can construct an infinite computation tree not leaving its own class.

[1] The equivalence defined in this section was the result of collaborative work with Sjoerd Cranen. The contributions of the author of this thesis are the definition of governed stuttering bisimulation and its decidability, as included in this thesis. Alternative characterisations, the equivalence proof, and the proof that governed stuttering bisimilarity refines winner equivalence will be part of Sjoerd Cranen's forthcoming PhD thesis. The results can also be found in [CKW12b; CKW12a].

The notation introduced in Section 4.1 allows us to adapt the definition of stuttering equivalence from computation paths to computation trees. We here provide the symmetric definition that we presented in [CKW12b], in which some alternative definitions are described as well. This version of the definition nicely facilitates the presentation of the algorithm.

**Definition 4.37.** Let $R \subseteq V \times V$ and $v, v' \in V$. Then $R$ is a governed stuttering bisimulation iff $R$ is an equivalence relation and $v \mathrel{R} v'$ implies:

a) $\Omega(v) = \Omega(v')$;

b) $v \,{}_i\!\!\mapsto_R \mathscr{C}$ iff $v' \,{}_i\!\!\mapsto_R \mathscr{C}$ for all $i \in \{\Diamond, \Box\}$, $\mathscr{C} \in V_{/R} \setminus \{[v]_R\}$;

c) $v \,{}_i\!\!\mapsto_R$ iff $v' \,{}_i\!\!\mapsto_R$ for all $i \in \{\Diamond, \Box\}$.

In this definition, observe that we have dropped the restriction on players in clause $a$), and instead in clause $b$) we require that, if from vertex $v$ player $i$ can force play to class $\mathscr{C}$, then from $v'$ the same player must also have a strategy to force the game to this class, regardless of the moves of the opponent. In clause $c$) we require for both players that, if the player can force the game to diverge from vertex $v$, then he must also be able to do so from related vertices $v'$. This last restriction precisely allows us to circumvent the problem illustrated in Example 4.36.

### 4.5.1 Properties of Governed Stuttering Equivalence

Next we repeat the key properties of governed stuttering bisimulation that were described in [CKW12b].

First, observe that, if in Definition 4.37 we additionally require that $\mathscr{P}(v) = \mathscr{P}(v')$, we find that $v \mapsto_R U$ iff $v \,{}_{\mathscr{P}(v)}\!\!\mapsto_R U$, and, likewise, $v \mapsto_R$ iff $v \,{}_{\mathscr{P}(v)}\!\!\mapsto_R$. This is the basis for the following proposition, which shows that indeed governed stuttering bisimulation is coarser than stuttering equivalence.

**Proposition 4.38.** *Let $R \subseteq V \times V$ be a governed stuttering bisimulation, such that $v \mathrel{R} v'$ implies $\mathscr{P}(v) = \mathscr{P}(v')$. Then $R$ is a stuttering bisimulation.*

Like the other relations we have studied, governed stuttering bisimulation is an equivalence relation, as is witnessed by the following theorem.

**Theorem 4.39.** *$\simeq$ is an equivalence relation.*

Proving that $\simeq$ is an equivalence relation on parity games is far from straightforward: transitivity no longer bows to the standard proof strategies that work for stuttering bisimilarity and branching bisimilarity [GW96; Bas96]. Proving that the equivalence closure of the union of two governed stuttering bisimulation relations is again a governed stuttering bisimulation relation is equally problematic. For the details of the equivalence proof, which follows [Bas96] on a high level, the reader is referred to [CKW12b].

Not all equivalence relations admit a quotienting operation; in particular, delayed simulation [FW06] fails to have a natural quotienting operation. In contrast to this, the quotient for governed stuttering bisimulation can be defined.

Like we have seen in the quotient for governed bisimulation, the governed stuttering bisimulation quotient allows some freedom in choosing the player for the vertex representing a class that contains vertices owned by different players. In this definition, we furthermore need to take computation trees into account. We therefore first define a notion of minimality, and subsequently use this to define our quotient.

**Definition 4.40** (Minimality)**.** A minimal governed stuttering bisimilar representation of a parity game $(V, \to, \Omega, \mathscr{P})$ is defined as a game $(V_m, \to_m, \Omega_m, \mathscr{P}_m)$, that satisfies the following conditions (where $c, c', c'' \in V_m$):

$$
\begin{aligned}
V_m &= \{\, [v]_{\simeq} \mid v \in V \,\} \\
\Omega_m(c) &= \Omega(v) \text{ for all } v \in c \\
\mathscr{P}_m(c) &= i, \text{ if for all } v \in c, \text{ and some } c' \neq c \text{ we have } v \,{}_i{\mapsto}_{\simeq}\, c' \text{ and } v \,{}_{\neg i}{\not\to}_{\simeq}\, V \setminus c' \\
c \to_m c &\text{ iff } v \,{}_i{\mapsto}_{\simeq} \text{ for all } v \in c \text{ for some player } i \\
c \to_m c' &\text{ iff } v \,{}_i{\mapsto}_{\simeq} c' \text{ for all } v \in c \text{ for some player } i \text{ and } c' \neq c
\end{aligned}
$$

The third clause above leaves some freedom in defining the quotient. If from some vertex $v$ the play could be forced to $c'$ by $i$ without $\neg i$ having the opportunity to diverge, player $i$ is in charge of the game when the play arrives in $c$. This requires the representative in the quotient to be owned by player $i$. In all other cases, none of the players—or both of the players, depending on your point of view—is in charge of the game when the play arrives in $c$, hence the player that we choose is irrelevant. As a consequence a parity game may have multiple $\simeq$-minimal representations. Observe that every parity game contains at least as many vertices and edges as its $\simeq$-minimal representations, and $\simeq$-minimal representations are equal, except for the players that are assigned to the vertices. Moreover, any parity game is governed stuttering bisimulation equivalent to all its $\simeq$-minimal representations. The intricacies in quotienting governed stuttering bisimulation, and the existence of multiple minimal representations of parity games are illustrated in the following example.

**Example 4.41.** Consider the parity game in Figure 4.2a. Two of its four minimal representations are in Figure 4.2b and 4.2c. Observe that the particular player chosen for the 0 and 1 vertices is arbitrary and does not impact the solution to the games.

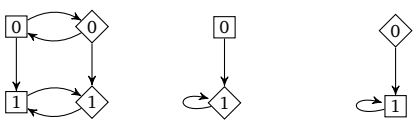



Figure 4.2: Both (b) and (c) are minimal representations of (a).

We define the $\simeq$-quotient of a parity game as the least $\simeq$-minimal representation according to the following ordering. For $\simeq$-minimal representation we define an ordering $\lll$ as follows. Let $\mathscr{G} = (V, \to, \Omega, \mathscr{P})$ and $\mathscr{G}' = (V', \to', \Omega', \mathscr{P}')$ be $\simeq$-minimal representations of the same parity game. We say that $\mathscr{G} \lll \mathscr{G}'$, if for all $v \in V$ and $v' \in V'$ such

that $v \simeq v'$, it is the case that $\mathscr{P}(v) \lessdot \mathscr{P}(v')$, where $\lessdot$ is the ordering on players from Section 4.3. To $\lll$, we associate the minimum $\min_{\lll}$. This finally allows us to define a unique governed stuttering bisimulation quotient.

**Definition 4.42** (Quotient)**.** Let $\mathscr{G}$ be a parity game. Its governed stuttering bisimulation quotient $\mathscr{G}'$ is defined as $\min_{\lll}\{\mathscr{G}_m \mid \mathscr{G}_m \text{ is a } \simeq\text{-minimal representation of } \mathscr{G}\}$.

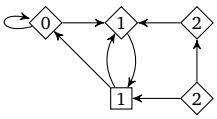
Governed stuttering bisimulation refines winner equivalence, as illustrated by the following proposition, and therefore we are allowed to reduce parity games using governed stuttering bisimulation prior to solving.

**Proposition 4.43.** *Governed stuttering bisimulation refines winner equivalence.*

This proposition can be proven along the same lines as Theorem 4.31, *i.e.*, for $v \simeq w$, and a strategy $\varphi$ from $v$ we construct a corresponding strategy $\psi$ from $w$. In this case however, the proof is more involved, since we have to ensure that *computation trees* can be mimicked, instead of *computation paths*. For details of the proof see [CKW12b].

It should not come as a surprise that governed stuttering bisimulation is able to relate vertices that cannot be related using stuttering bisimulation or governed bisimulation.

**Example 4.44.** Consider the parity game depicted below.




The equivalence relation that relates vertices with equal priorities is a governed stuttering bisimulation. Stuttering bisimulation and governed bisimulation do not relate any of the vertices. Winner equivalence relates the vertex with priority 0 to no other vertex, and all other vertices are related.

We summarise the relationship with other equivalences in the following theorem.

**Theorem 4.45.** *Governed stuttering equivalence*

- *is strictly refined by stuttering equivalence, i.e.,* $\simeq \subseteq \approx$*;*
- *is strictly refined by governed bisimulation, i.e.,* $\leftrightarrows \subseteq \approx$*;*
- *strictly refines winner equivalence, i.e.,* $\simeq \subseteq \sim_w$*;*
- *is incomparable to direct simulation, delayed simulation, and the biased variations of delayed simulation.*

*Proof.* Refinement by stuttering bisimulation and governed bisimulation follows immediately from the definitions. The fact that governed stuttering bisimulation refines winner equivalence is a consequence of Proposition 4.43. Strictness in all three cases follows from the parity game given in Example 4.44. Incomparability with direct simulation follows from Example 4.34. □

### 4.5.2 Decidability

We present an algorithm for deciding governed stuttering bisimilarity inspired by Groote and Vaandrager's $\mathcal{O}(nm)$ algorithm for deciding stuttering bisimilarity [GV90]. Before we provide the details, we introduce the necessary additional concepts.

The algorithm follows a partition refinement approach. A *partition* $P$ of a set of vertices $V$ is a set of non-empty, disjoint subsets of $V$. Formally, $P = \{B \mid B \subseteq V\}$, where $\bigcup_{B \in P} = V$, and for all $B, B' \in P$, it holds that $B \cap B' \neq \emptyset \implies B = B'$. Furthermore $\emptyset \notin P$. We refer to the elements of a partition as *blocks*. A partition $P$ induces a relation in which vertices are related if and only if they are in the same block. By abuse of notation we refer to this relation as $P$ as well.

Our algorithm requires a generalisation of the notion of *attractor sets* [McN93] along the lines of the generalisation used for the computation of the *Until* in the alternating-time temporal logic ATL [AHK02]. The generalisation introduces a parameter restricting the set of vertices that are considered in the attractor sets.

**Definition 4.46.** Let $(V, \to, \Omega, \mathscr{P})$ be a parity game, let $B, U \subseteq V$, and let $i$ be a player. The attractor set for player $i$ into $U$, extended only with vertices in $B$, is defined inductively as follows.

$$\begin{array}{lcl} {}_B Attr_i^0(U) & \triangleq & U \\ {}_B Attr_i^{n+1}(U) & \triangleq & {}_B Attr_i^n(U) \\ & \cup & \{v \in B \mid \mathscr{P}(v) = i \wedge \exists v \to v' : v' \in {}_B Attr_i^n(U)\} \\ & \cup & \{v \in B \mid \mathscr{P}(v) \neq i \wedge \forall v \to v' : v' \in {}_B Attr_i^n(U)\} \\ {}_B Attr_i(U) & \triangleq & {}_B Attr_i^\omega(U) \end{array}$$

Observe that this definition satisfies the following property.

*Property* 4.47. Let $(V, \to, \Omega, \mathscr{P})$ be a parity game, let $B, U \subseteq V$, and let $i$ be a player, then $V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B)) = {}_B Attr_{\neg i}(V \setminus B)$.

The set $Leave_i(B, U)$ captures the vertices of $B$ from which player $i$ can *leave* $B$, and move to $U$.

**Definition 4.48.** Let $(V, \to, \Omega, \mathscr{P})$ be a parity game, let $B, U \subseteq V$, and let $i$ be a player. The subset of $B$ from which player $i$ can force the game to $U$ is defined as follows.

$$Leave_i(B, U) \triangleq {}_B Attr_i(U) \cap B$$

We use *Leave* as an algorithmic characterisation of ${}_i\mapsto$. Their correspondence is formalised in Propositions 4.50 and 4.53 below; this allows for restating the criteria from Definition 4.37 in terms of *Leave*. We first capture some general properties about *Attr* and *Leave*.

The first lemma describes that from some vertices player $i$ can evade the attractor set, and that from some vertices player $\neg i$ cannot reach the attractor set.

**Lemma 4.49.** *Let $P$ partition $V$, and let $B \in P$ be a block. Define $U = B \setminus {}_B Attr_{\neg i}(V \setminus B)$, then:*

1. $\forall u \in U : \mathscr{P}(u) = i \implies (\exists u \to u' : u' \in U)$

2. $\forall u \in U : \mathscr{P}(u) \neq i \implies (\forall u \to u' : u' \in U)$

*Proof.* We prove the two properties separately.

1. Let $u \in U$ and $\mathscr{P}(u) = i$. Towards a contradiction, suppose that $\forall u \to u'$ it holds that $u' \notin U$. Because of totality of $\to$ there must be at least one such $u'$. It follows that $\forall u \to u' : u' \in V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B))$. Hence $u' \in {}_B Attr_{\neg i}(V \setminus B)$, for all $u'$ such that $u \to u'$, according to Property 4.47. By definition of *Attr*, then also $u \in {}_B Attr_{\neg i}(V \setminus B)$, but then $u \notin B \setminus {}_B Attr_{\neg i}(V \setminus B)$, which contradicts $u \in U$, hence $(\exists u \to u' : u' \in U)$.

2. Let $u \in U$ and $\mathscr{P}(u) \neq i$. Towards a contradiction, suppose that $\exists u \to u'$ such that $u' \notin U$. Let $u'$ be such. Observe that $u' \in V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B))$, and thus $u' \in {}_B Attr_{\neg i}(V \setminus B)$. By definition of *Attr*, then also $u \in {}_B Attr_{\neg i}(V \setminus B)$, thus $u \notin U$, which contradicts the assumption that $u \in U$, hence $\forall u \to u' : u' \in U$. □

The computation of *Leave* can be used to establish the divergence criterion for all vertices in a block, as formalised by the following proposition.

**Proposition 4.50.** *Let $P$ be a partition of $V$, and let $B \in P$ be a block. Then for all $u \in B$: $u \in Leave_{\neg i}(B, V \setminus B)$ if and only if $u \; {}_i\not\to_P$.*

This proposition follows immediately from the following two lemmata.

**Lemma 4.51.** *Let $P$ partition $V$, and let $B \in P$ be a block. Then for all $u \in B$ : $u \in Leave_{\neg i}(B, V \setminus B) \implies u \; {}_i\not\to_P$.*

*Proof.* Let $P$ partition $V$, and let $B \in P$ be a block, and suppose that $u \in Leave_{\neg i}(B, V \setminus B)$. By definition of *Leave*, $Leave_{\neg i}(B, V \setminus B) = {}_B Attr_{\neg i}(V \setminus B) \cap B$. Observe that, if $u \in {}_B Attr_{\neg i}(V \setminus B) \cap B$, there is some least $n$ such that $u \in {}_B Attr^n_{\neg i}(V \setminus B) \cap B$. We show by induction on $n$ that $\forall u \in B : u \in {}_B Attr^n_{\neg i}(V \setminus B) \cap B \implies u \; {}_i\not\to_P$.

- $n = 0$. We find that ${}_B Attr^0_{\neg i}(V \setminus B) = V \setminus B$, and $(V \setminus B) \cap B = \emptyset$, hence the statement vacuously holds.

- $n = m + 1$. As induction hypothesis assume that $\forall u \in B : u \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies u \; {}_i\not\to_P$.

  Let $u \in {}_B Attr^{m+1}_{\neg i}(V \setminus B) \cap B$. By definition of *Attr*, we find three cases.

  - $u \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B$. This follows immediately from the induction hypothesis.
  - $u \in \{v \in B \mid \mathscr{P}(v) \neq i \wedge \exists v \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)\} \cap B$. So we know that $\mathscr{P}(u) \neq i$, and $\exists u \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)$. Let $u'$ be such, and observe that either $u' \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B$ or $u' \in V \setminus B$ because $(V \setminus B) \subseteq {}_B Attr^m_{\neg i}(V \setminus B)$. In the latter case there is a strategy for player $\neg i$ such that $u \; {}_{\neg i}\!\!\mapsto_P V \setminus B$, and by Lemma 4.4 we find $u \; {}_i\not\to_P$. In the first case, observe that by the induction hypothesis, $u' \; {}_i\not\to_P$. According Lemma 4.4, we find that $u' \; {}_{\neg i}\!\!\mapsto_P V \setminus B$. Application of Lemma 4.5 gives us that $u \; {}_{\neg i}\!\!\mapsto_P V \setminus B$, and again using Lemma 4.4, we have $u \; {}_i\not\to_P$.

– $u \in \{v \in B \mid \mathscr{P}(v) = i \wedge \forall v \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)\} \cap B$. So we know that $\mathscr{P}(u) = i$, and $\forall u \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)$. By totality of $\to$ there is at least one such $v'$. Observe that $u \,{}_{\neg i}\!\!\mapsto_P {}_B Attr^m_{\neg i}(V \setminus B)$. According to the induction hypothesis, we know $\forall v \in B : v \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies v \,{}_i\!\!\not\mapsto_P$. Applying Lemma 4.4 we also know that $\forall v \in B : v \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies v \,{}_{\neg i}\!\!\mapsto_P (V \setminus B)$. Using Lemma 4.5 we find that $u \,{}_{\neg i}\!\!\mapsto_P (V \setminus B)$. Another application of Lemma 4.4 gives us the desired result $u \,{}_i\!\!\not\mapsto_P$. □

**Lemma 4.52.** *Let $P$ be a partition of $V$, and let $B \in P$ be a block. Then for all $u \in B : u \notin Leave_{\neg i}(B, V \setminus B) \implies u \,{}_i\!\!\mapsto_P$.*

*Proof.* Define $U \subseteq B$ to be the subset of $B$ that cannot be forced by player $\neg i$ to leave $B$, *i.e.* $U = B \setminus Leave_{\neg i}(B, V \setminus B) = B \setminus {}_B Attr_{\neg i}(V \setminus B)$. Observe that for all $u \in V : u \in U$ iff $u \notin Leave_{\neg i}(B, V \setminus B)$, so we reformulate our goal as $\forall u \in B : u \in U \implies u \,{}_i\!\!\mapsto_P$. We define strategy $\sigma \in \mathbb{S}_i$ that is defined for vertices in $U$, such that

$$\sigma(u) = \sqcap\{u' \in U \mid u \to u'\}$$

Observe that $\{u' \in U \mid u \to u'\} \neq \emptyset$ due to Lemma 4.49, hence $\sigma(u)$ is well-defined. Furthermore for all $v \in U$ with $\mathscr{P}(v) \neq i$, all successors are in $U$ due to Lemma 4.49. As a result, $u \,{}_\sigma\!\!\mapsto_P$, and hence $u \,{}_i\!\!\mapsto_P$. □

Now we can establish whether all vertices in a block diverge by consider *Leave*. It remains to decide what vertices in a block can reach some other block. This can be decided in a similar way.

**Proposition 4.53.** *Let $P$ partition $V$, and let $B, B' \in P$ such that $B \neq B'$. Let $v \in B$ such that $v \to B'$. Then for all $w \in B$ it holds that $w \,{}_{\mathscr{P}(v)}\!\!\mapsto_P B'$ if and only if $w \in Leave_{\mathscr{P}(v)}(B, B')$.*

*Proof.* The proof of this lemma follows the same line of reasoning as the proof of Proposition 4.50. □

Using the correspondence proven in these propositions we can redefine governed stuttering equivalence in terms of *Leave*. This results in the following characterisation of governed stuttering bisimulation that is more readily converted into an algorithm.

**Theorem 4.54.** *Let $P$ partition $V$, and let $v, v' \in V$. $P$ is a governed stuttering bisimulation if and only if $v \; P \; v'$ implies:*

*a)* $\Omega(v) = \Omega(v')$;

*b)* $v \in Leave_{\mathscr{P}(v)}(B, \mathscr{C})$ *iff* $v' \in Leave_{\mathscr{P}(v)}(B, \mathscr{C})$, *where* $v, v' \in B$, $B, \mathscr{C} \in P$, *and* $B \neq \mathscr{C}$;

*c)* $v \notin Leave_{\neg i}(B, V \setminus B)$ *iff* $v' \notin Leave_{\neg i}(B, V \setminus B)$ *for all* $i \in \{\Diamond, \Box\}$, *and for* $v, v' \in B$ *with* $B \in P$.

*Proof.* The first requirement has been left unchanged. The second requirement has been reformulated according to Proposition 4.53, and the last requirement has been expressed in terms of *Leave* according to Proposition 4.50. □

Groote and Vaandrager's algorithm for stuttering bisimulation repeatedly refines a carefully chosen initial partition $P_0$ using a so-called *splitter*. We apply the same principle, choosing $P_0$ such that for all $v, v' \in V$, $v\ P_0\ v'$ if and only if $\Omega(v) = \Omega(v')$ as our initial partition. As our splitter, we define a function *pos* that returns the set of vertices in $B$ from which a given player $i$ can force the play to reach $B'$, or, in case $B = B'$, force the play to diverge:

$$pos_i(B, B') = \begin{cases} Leave_i(B, B') & \text{if } B \neq B' \\ Leave_{\neg i}(B, V \setminus B) & \text{if } B = B' \end{cases}$$

In line with [GV90], we say that $B'$ is a *splitter* of $B$ if and only if $\emptyset \neq pos_i(B, B') \neq B$ for some player $i$. A partition $P$ is *stable with respect to a block* $B \in P$ if $B$ is not a splitter of any block in $P$. The partition itself is stable if it is stable with respect to all its blocks.

A high-level description of our algorithm for governed stuttering bisimulation, is given as Algorithm 1. The current partitioning is repeatedly refined until it stabilises. In a refinement step, the splitter is initially undefined (denoted $\bot$), we then determine, for each block $B$, whether a splitter for $B$ exists. If a splitter is found, $B$ is split into two new blocks, and we iterate.

**Algorithm 1** Decision procedure for $\simeq$ (high-level view)

$n \leftarrow 0$
**repeat**
  $splitter \leftarrow \bot$
  **for each** $B \in P_n$ and player $i$ **do** { Find splitter in $\mathcal{O}(nm)$ }
    **if** there exists $v \in B$ with $v \to B'$ and $\emptyset \neq pos_i(B, B') \neq B$ for $B' \in P_n$ **then**
      $splitter \leftarrow (B, pos_i(B, B'))$
    **end if**
  **end for**
  **if** $splitter = (B, Pos)$ **then** { Refine partition in $\mathcal{O}(m)$ }
    $P_{n+1} \leftarrow (P_n \setminus \{B\}) \cup \{Pos, B \setminus Pos\}$
  **end if**
  $n \leftarrow n + 1$
**until** $P_{n-1} = P_n$

Algorithm 1 merely gives a conceptual overview of the partition refinement algorithm. However, an implementation of this algorithm is not efficient, since it considers too many blocks. We now optimise the algorithm using the approach from [GV90].

Given the nature of the definition of divergence in governed stuttering bisimulation, we cannot remove divergent states in a preprocessing step as is done in [GV90]. Note that this preprocessing is not necessary, even in the original algorithm. This was shown by Vu [Vu07], who modified the algorithm to decide orthogonal bisimulation. In *ibid.*, the algorithm does not perform a preprocessing for divergent states, yet the algorithm still runs in $\mathcal{O}(nm)$ time.

The algorithm we describe maintains a partition $P$, which is initially chosen to be $P = P_0$. Algorithm 2 then computes the largest governed stuttering bisimulation. For every block $B$, the incoming edges from another block are recorded in $B.incoming$.

**Algorithm 2** Algorithm for computing $\simeq$ (detailed view)

```
todo ← P_0; stable ← ∅;
repeat
  B' ← head(todo)
  for each v → v' ∈ B'.incoming do
    if v ∉ B then
      BL.append(v)
    end if
  end for
  repeat
    B ← BL.pop() {Determine for each B whether B' is a splitter for B}
    (foundsplitter, inert_becomes_non_inert, B_1, B_2) ← TrySplit(B, B')
  until foundsplitter or BL = ∅
  if foundsplitter then
    todo.remove(B)
    todo.append(B_1, B_2)
    if inert_becomes_non_inert then
      todo ← todo + stable; stable ← ∅
    end if
  else
    todo.remove(B')
    stable.append(B')
  end if
until todo = ∅
```

Given a parity game, our algorithm maintains two lists of blocks, *todo* and *stable*. A block $B$ is in *stable* if the current partition is known to be stable with respect to $B$, otherwise $B$ is in *todo*. Initially all blocks in $P_0$ are in *todo*. While the todo list is not empty, we check for each block $B'$ in the list whether it is a splitter of any block $B$ using the routine $\mathit{TrySplit}(B, B')$. If a splitter is found, the todo list is updated accordingly, and if some inert transition has become non-inert, *i.e.*, a transition that was internal to a block now points to a different block, all stable blocks are added to the todo list in accordance with Lemma 4.56 on page 99.

We next elaborate on $\mathit{TrySplit}(B, B')$, which is given as Algorithm 3. This routine determines whether $B'$ is a splitter of $B$ using *Leave*. If a splitter is found, the actual splitting is performed; the non-inert edges are added to the appropriate block, and for all inert edges it is checked whether they are still inert, and if not they are also added as non-inert edges to the appropriate block.

Our algorithm indeed computes $\simeq$ as witnessed by the following lemma.

**Lemma 4.55.** *If $P$ is a stable partition refining $P_0$, then $P$ is a governed stuttering bisimulation. If $P$ is the largest such partition then $P$ coincides with $\simeq$.*

*Proof.* We first prove the first part of the statement. Let $P$ be a stable partition refining $P_0$, and let $v, v' \in V$ such that $v \; P \; v'$. We show that $P$ satisfies the three properties described in Definition 4.37. $\Omega(v) = \Omega(v')$ follows immediately as $P$ is a refinement of

**Algorithm 3** *TrySplit*$(B, B')$

```
foundsplitter ← false; inert_becomes_non_inert ← false
i ← 0
repeat
  B_1 ← pos_i(B, B')
  i ← i + 1
  foundsplitter ← ∅ ≠ B_1 ≠ B
until i > 1 or foundsplitter
if foundsplitter then
  B_2 ← B \ B_1
  for v → v' ∈ B.incoming do
    if v' ∈ B_1 then
      B_1.incoming.append(v → v')
    else
      B_2.incoming.append(v → v')
    end if
  end for
  for C ∈ {B_1, B_2} do
    for v ∈ C do
      for v' → v ∈ v.incoming do
        if v' ∉ C then
          inert_becomes_non_inert ← true
          v.incoming.remove(v' → v)
          C.incoming.append(v' → v)
        end if
      end for
    end for
  end for
  return (foundsplitter, inert_becomes_non_inert, B_1, B_2)
else
  return (foundsplitter, inert_becomes_non_inert, ∅, ∅)
end if
```

$P_0$. The transfer and divergence properties follow immediately from the observation that $P$ is stable, the definition of *pos* and Theorem 4.54.

For the second part of the statement observe that $\simeq$ is an equivalence relation according to Theorem 4.39. Furthermore it is defined to be the largest governed stuttering bisimulation, and it induces a stable refinement of $P_0$. As any stable refinement is a governed stuttering bisimulation our result follows. □

Algorithm 2 relies on the observation that stable blocks only have to be reconsidered if inert transitions become non-inert, which is shown in the following lemma.

**Lemma 4.56.** *Let $P_n$ and $P_{n+1}$ be partitions of $V$, such that $P_n$ and $P_{n+1}$ have the same inert transitions. Assume that $P_{n+1}$ refines $P_n$. Let $B' \in P_n, P_{n+1}$ such that $P_n$ is stable with respect to $B'$. Then also $P_{n+1}$ is stable with respect to $B'$.*

*Proof.* Towards a contradiction, suppose that there exists a block $B \in P_{n+1}$ such that $B' \in P_n, P_{n+1}$ is a splitter of $B$, we show that than $P_n$ is not stable with respect to $B'$. We distinguish two cases.

1. $B = B'$. As $B'$ is a splitter of itself under $P_{n+1}$, and $B' \in P_n$, we immediately find that $B'$ is a splitter of itself under $P_n$, which is a contradiction.

2. $B \neq B'$. As $B'$ is a splitter of $B$, we know that $v, v' \in B$ such that $v \;{}_i\!\!\mapsto_{P_{n+1}} B' \wedge v' \;{}_i\!\!\not\mapsto_{P_{n+1}} B'$ for some player $i$. Let $v, v'$ be such. Observe that there is a block $B'' \in P_n$ such that $B \subseteq B''$. In case $B = B''$, then immediately we find that $B'$ is a splitter of $B''$, which violates the stability of $P_n$. Suppose $B \subset B''$. As $P_n$ and $P_{n+1}$ have the same inert transitions, there are no $u \in B$, $u' \in B''$ such that $u \to u'$. From this, and the assumption that $v' \;{}_i\!\!\not\mapsto_{P_{n+1}} B'$ it follows that $v' \;{}_i\!\!\not\mapsto_{P_n} B'$. Likewise we have that $v \;{}_i\!\!\mapsto_{P_n} B'$, and hence $B'$ is a splitter of $B''$ under $P_n$, which contradicts stability of $P_n$.

In both cases we find a contradiction, hence $P_{n+1}$ is stable with respect to $B'$. □

The number of iterations the algorithm requires to compute $\simeq$ is bounded in the following theorem.

**Theorem 4.57.** *Algorithm 2 terminates after at most $n-|P_0|$ refinement steps. The resulting partition $P_f$ is the coarsest stable partition refining $P_0$.*

*Proof.* Termination of the algorithm after at most $n - |P_0|$ refinement steps is straightforward. Next we show that the resulting partition $P_f$ is the coarsest stable partition refining $P_0$. We prove, by induction on the number of refinement steps $j$, that any stable partition refining $P_0$ is also a refinement of the current partition $P_j$. Clearly the statement holds initially. Let $R$ be a stable refinement of $P_0$. By the induction hypothesis, $R$ is a refinement of $P_j$. Let $P_{j+1}$ be the refinement of $P_j$, after refining using splitting pair $(B, B')$ (*i.e.*, $B'$ is a splitter of $B$). We show that $R$ is also a refinement of $P_{j+1}$. Let $C$ be a block in $R$. Then there is a block $D$ in $P_j$ such that $C \subseteq D$. We show that $C$ is included in a block of $P_{j+1}$. In case $D \neq B$ we are done. In case $D = B$, assume that splitting was done with respect to player $i$, then we have to show that either $C \subseteq pos_i(B, B')$, or $C \subseteq B \setminus pos_i(B, B')$.

Towards a contradiction, suppose that there are $v, v' \in C$ such that $v \in pos_i(B, B')$ and $v' \notin pos_i(B, B')$. We distinguish two cases:

- $B \neq B'$. As $v \in pos_i(B, B')$, we know $v \;{}_i\!\!\mapsto_{P_j} B'$. We know there is a sequence of classes $C_1, \ldots, C_n$ in $R$, with $C_1 = C$, $C_n = C'$, and $C_j \;{}_i\!\!\to C_{j+1}$, in other words, in each of the classes $C_j$, the game can be forced to $C_{j+1}$ by player $i$. Since $R$ is a stable refinement of $P_0$, and $v, v' \in C$ we find that also $v' \in pos_i(B, B')$, which is a contradiction.

- $B = B'$. As $v \in pos_i(B, B')$, we know $v \;{}_i\!\!\mapsto_{P_j}$. We know there is a sequence of classes $C_1, \ldots, C_n$ in $R$ such that $C_j \;{}_i\!\!\mapsto_R C_{j+1}$, and $C = C_1 = C_n$. Following a similar line of reasoning as in the previous case, we find that then $v' \in pos_i(B, B')$, which is a contradiction.

□

Given the above considerations, we can determine the running time complexity of our algorithm as follows.

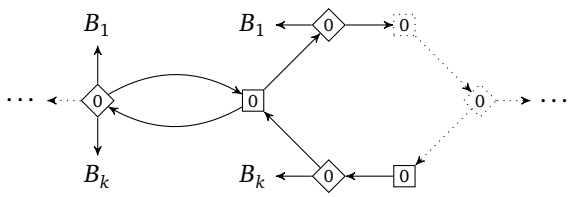
**Theorem 4.58.** *Algorithm 1 decides $\simeq$ in $\mathcal{O}(n^2m)$ time for a parity game that contains $n$ vertices and $m$ edges.*

*Proof.* Let $n_B$ and $m_B$ be the number of vertices and the number edges in block $B$. Deciding whether a block $B'$ is a splitter of block $B$ in Algorithm 3 takes $\mathcal{O}(m_B + n_B)$ time (the time required to compute the attractor set). If a splitter is found, the actual splitting takes $\mathcal{O}(m)$ time. As a result, deciding whether a block $B'$ is a splitter for the current partition takes $\Sigma_B(\mathcal{O}(m_B + n_B)) = \mathcal{O}(m)$ time. Finding a splitter of the current partition (if it exists) hence has time complexity $\mathcal{O}(nm)$. As only $n - |P_0|$ refinement steps are possible (Lemma 4.57), the time complexity of $\mathcal{O}(n^2m)$ follows. □

Our time complexity is worse than the $\mathcal{O}(nm)$ achieved by the original algorithm for deciding stuttering bisimulation. The extra factor $\mathcal{O}(n)$ is due to the complexity required to search for a splitter which, in our case, requires $\mathcal{O}(nm)$ time, instead of the original $\mathcal{O}(m)$ time.

Investigating the origins of this extra factor of the number of vertices in searching a splitter, and carefully analysing all cases, uncovers a single problematic case: finding a splitter for a block $B$ that consists of a single strongly connected component in which all vertices in $B$ are divergent for exactly one player. For this case, we only have an $\mathcal{O}(nm_C)$ algorithm, leading to the $\mathcal{O}(nm)$ time bound for a single iteration. The following problem statement formalises this 'hard case'.

**Problem 4.59.** Let $P$ be a partition, and let $C \in P$ be a block, such that $C$ is a strongly connected component, and for all $v \in C$ we have $v \,{}_i\!\!\mapsto_P$, and $v \,{}_{\neg i}\!\!\not\mapsto_P$. Determine, in $\mathcal{O}(m)$ time, whether there exist $v, v' \in C$ and a block $B \in P$ such that $v \,{}_i\!\!\mapsto_P B$ and $v' \,{}_i\!\!\not\mapsto_P B$.




In the situation sketched above, containing a single SCC with $2k + 1$ vertices, in which player $\diamond$ can enforce divergence, whereas player $\square$ cannot, we need $\mathcal{O}(km_C)$ to determine that there is no splitter for this block, where $m_C$ is the number of edges in the SCC. To achieve the desired running time bounds, we should be able to solve this in $\mathcal{O}(m_C)$ time instead.

**Claim 4.60.** *Given a solution to Problem 4.59, the largest governed stuttering bisimulation can be computed in $\mathcal{O}(nm)$ time.*

Given that the problematic case is extremely specific, we hypothesise that, in practice, this case does not occur often, and that the algorithm for governed stuttering bisimulation will be competitive with that for stuttering equivalence. This hypothesis is tested in the next chapter.

## 4.6 Closing Remarks

In this chapter we have restated the equivalences from Chapter 3 in terms of parity games. We investigated weaker equivalences, and their applicability to parity games. In particular, we modified stuttering equivalence so that it can be used to reduce parity games, and proved that stuttering bisimilar vertices in a parity game have the same winner. Actually, our result is even stronger: given a strategy in one parity game, a mimicking strategy in a stuttering bisimilar parity game can be constructed.

The ideas of governed bisimulation and stuttering equivalence were combined into a novel relation, dubbed *governed stuttering bisimulation*, in which vertices owned by different players may be related. We showed that this relation is an equivalence relation that can be decided in $\mathcal{O}(n^2m)$ time using a partition refinement algorithm. This time complexity is worse than the $\mathcal{O}(nm)$ time complexity for deciding stuttering bisimulation, but we expect that in practice the algorithm will be largely competitive with the one for stuttering bisimilarity, since the worst case is mainly due to a single degenerate case.

We have not proven a lower bound on the running time complexity of our algorithm. This leaves the question open whether our algorithm decides governed stuttering equivalence in $\mathcal{O}(nm)$ time, instead of the $\mathcal{O}(n^2m)$ that we have proven. There are two alternative formulations of this problem that can be investigated to answer this question. First, towards a positive answer, the analysis can be improved to show that governed stuttering bisimulation can be decided in $\mathcal{O}(nm)$ time. This most likely involves solving Problem 4.59. Second, towards a negative answer, a class of examples must be provided for which our algorithm indeed exhibits an $\mathcal{O}(n^2m)$ worst case running time.

In this chapter we have shown the relations between all equivalences that we have defined, as well as the relation between our equivalences and direct simulation and Fritz and Wilke's delayed simulation [FW06]. The corresponding lattice was presented in the introduction of this chapter.

An obvious question is whether elements of delayed simulation and governed stuttering bisimulation can be combined. Given the complexity of the proofs of most of our results for governed stuttering bisimulation and our attempts to weakening governed stuttering bisimulation along these lines, we are rather sceptic about the chances of success. Even if one would manage to define such a relation, it would likely have little practical significance due to the prohibitive complexity, $\mathcal{O}(n^3d^2m)$, of delayed simulation, where $n$, $m$ and $d$ are the number of vertices, edges and priorities in the game, respectively.

An interesting extension of our work could be to generalise the concepts of governed stuttering bisimilarity to games with other winning conditions that are insensitive to stuttering such as Muller, Rabin and Streett conditions. We expect such a generalisation to be reasonably straightforward, as long as the winning conditions only reason about the infinitely often recurring priorities.

Finally, we observe that stuttering bisimulation (also known as *branching bisimulation* in labelled transition systems) underlies several confluence reduction techniques for syntactic system descriptions, see *e.g.* [GS95; GS96; GP00; BP02]. Such reductions partly side-step the state-space explosion problem. We believe that our study offers the required foundations for bringing similar-spirited confluence reduction techniques to a setting of symbolic representations of parity games.

# Chapter 5

# Benchmarking Parity Games

In the previous chapter we have introduced four solution-preserving reductions for parity games. So far we have not systematically studied their practical significance. In this chapter we study how well our equivalences are able to reduce the size of parity games that are created for typical verification problems. Furthermore, we investigate whether our reductions are helpful in reducing the time required for solving parity games, *i.e.*, is first reducing a parity game, and subsequently solving it, faster than outright solving the original parity game.

We would like to investigate the practical significance of our reductions using a standard set of benchmarks. Friedmann and Lange [FL09] observed in 2009 that no such standard benchmark set for parity games was available, and they introduced a small benchmark set. To the best of our knowledge, the situation has not improved since then, and their benchmarks still are the most comprehensive benchmarks observed in a single paper. The diversity of the parity games in this set is still very limited. Therefore, in this chapter we first develop a set of parity games for benchmarking. This benchmark set should be diverse, contain games that originate from different verification problems, and contain those games that have been used to experimentally evaluate algorithms in the literature.

A large number of parity game solving algorithms and related heuristics have been described in the literature, and some efforts have been made to compare them. Most notably, Friedmann and Lange evaluated the effect of heuristics in their generic solver. They considered parity games originating from decision procedures, two model checking problems, and random games [FL09]. In [KW11; CKW11; CKW12b] model checking problems and equivalence checking problems are used to evaluate the reducing capabilities of the equivalences they present. In general, parity game examples[1] in the literature can be classified in one of the following three classes:

1. Encodings of problems such as model checking, equivalence checking and complementation of Büchi automata to parity games. Such examples can be found in [Mad97; Mat03; PW08; Kei09; KW11; CKW11; CKW12b; FL09].

[1] In this thesis we also consider examples originally formulated as Boolean equation systems.

2. Parity games for which a certain solving algorithm requires exponential time, see, *e.g.*, [Mad97; Jur00; Obd06a; Fri09; FL10b; Fri11c; GW13].

3. Random games, of which a few different variations have been described, see [BV01; Lan05; Sch08a; Sch08b; FL09; FL10b].

Our benchmarks include examples from each of the categories described above.

The benchmarking setup is inspired by the explicit state model checking benchmarks described in [Pel06; Pel07]. These benchmarks were analysed with respect to a set of structural properties that was described in [Pel04]. In this chapter we first describe a similar set of structural properties for parity games, and we show that, with respect to these properties, our benchmark set is diverse. Note that we do not reuse the benchmarks from [Pel06; Pel07], because the properties that are considered are simple, leading to a set of parity games that is too restricted.

**Structure of this chapter.** We describe structural properties of parity games in Section 5.1. The implementation used for generating the benchmarks is briefly described in Section 5.3. The set of benchmarks is described in Section 5.2, of which we analyse the structure in Section 5.4. Subsequently, the effect of the equivalence relations described in Chapter 4 is studied in Section 5.5. We conclude this chapter in Section 5.6.

## 5.1 Structural Properties of Parity Games

Parity games are graphs in which the vertices are partitioned in two sets; therefore, structural properties of graphs can directly be used to characterise parity games. We describe a number of these properties in Section 5.1.1. The partitioning of vertices between two players, as well as the priorities in parity games, give rise to a number of more specific properties. These are described in Section 5.1.2.

### 5.1.1 Generic Graph Properties

We first describe a number of structural properties of graphs, that were also studied for state spaces in [Pel04; Pel06; Pel07]. In this section, let $(V, \rightarrow)$ be a graph.

**Basic Sizes.** As basic characteristics, we consider the number of vertices $|V|$, and the number of edges $|\rightarrow|$.

**Degrees.** Typical structural properties in the graph are the in- and out-degrees of vertices, *i.e.*, the number of incoming and outgoing edges of vertices. Formally, for vertex $v \in V$, the in-degree $\mathsf{indeg}(v)$ is defined as $|\{u \in V \mid u \rightarrow v\}|$, the out-degree $\mathsf{outdeg}(v)$ is $|\{w \in V \mid v \rightarrow w\}|$, and the degree $\mathsf{deg}(v)$ is $|\{w \in V \mid v \rightarrow w \vee w \rightarrow v\}|$. We record the minimum, maximum and average of these values.

The correspondence between in-degree, out-degree and the degree is characterised as $v \rightarrow v \implies \mathsf{deg}(v) = \mathsf{outdeg}(v) + \mathsf{indeg}(v) - 1$, and $v \not\rightarrow v \implies \mathsf{deg}(v) = \mathsf{outdeg}(v) + \mathsf{indeg}(v)$.

**Strongly Connected Components.** The strongly connected components (SCCs) of a graph are the maximal strongly connected subgraphs. More formally, a strongly connected component a maximal set $\mathscr{C} \subseteq V$ for which, for all $u, v \in \mathscr{C}$, $u \to^* v$, *i.e.*, each vertex in $\mathscr{C}$ can reach every other vertex in $C$.

The strongly connected components in a graph induce a quotient graph. Let $\mathsf{sccs}(V)$ denote the strongly connected components of the graph, then the quotient graph is the graph $(\mathsf{sccs}(V), \to')$ and for $\mathscr{C}_1, \mathscr{C}_2 \in \mathsf{sccs}(V)$, there is an edge $\mathscr{C}_1 \to' \mathscr{C}_2$ if and only if $\mathscr{C}_1 \neq \mathscr{C}_2$ and there exist $u \in \mathscr{C}_1$ and $v \in \mathscr{C}_2$ such that $u \to v$. Observe that the quotient graph is a directed acyclic graph.

We say that an SCC $\mathscr{C}$ is *trivial* if $|\mathscr{C}| = 1$ and $\mathscr{C} \not\to \mathscr{C}$, *i.e.*, it only contains one vertex and no edges, and we say that $\mathscr{C}$ is *terminal* if $\mathscr{C} \not\to'$, *i.e.*, its outdegree in the quotient graph is 0. The *SCC quotient height* of a graph is the length of the longest path in the quotient graph.

**Properties of Search Strategies.** Given some initial vertex $v_0 \in V$, breadth-first search (BFS) and depth-first search (DFS) are search strategies that can be used to systematically explore all vertices in the graph. The fundamental difference between BFS and DFS is that the BFS maintains a queue of vertices that still need to be processed, whereas the DFS maintains a stack of vertices. We record the queue and stack sizes during the search.

Breadth-first search induces a natural notion of levels, where a vertex is at level $k$ if it has least distance $k$ to $v_0$. The *BFS height* of a graph is $k$ if $k$ is the maximal non-empty level of the BFS. For each level the number of vertices at that level is recorded. During a BFS, three kinds of edges can be detected, *viz.* edges that go to a vertex that was not yet seen, edges that go to a vertex that was seen, but has not yet been processed (*i.e.*, vertices in the queue) and edges that go back to a vertex on a previous level. This last type of edges is also referred to as a *back-level edge*. Formally it is and edge $u \to v$ where the level of $u$, say $k_u$ is larger than the level of $v$, say $k_v$. The length of a back-level edge $u \to v$ is $k_u - k_v$.

**Distances.** The *diameter* of a graph is the maximal length of a shortest path between any pair of vertices. The *girth* is the length of the shortest cycle in the graph. Both measures basically require computing the all-sources-shortest path problem with unit weights on the edges, which is quadratic in the size of the graph.

For undirected graphs the diameter can be computed more efficiently using the techniques from Takes and Koster [TK11]. For directed graphs, however, no more efficient algorithm is known.

**Local Structure.** Pélanek also studied some local graph properties. A *diamond* rooted at a vertex $u$ is a quadruple $(u, v, v', w)$ such that $v \neq v'$, $u \to v$, $u \to v'$, $v \to w$, and $v' \to w$.

The *k-neighbourhood of v* is the set of vertices that can be reached from $v$ in at most $k$ steps (not counting $v$). The *k-clustering coefficient of v* is the ratio of the number of edges and the number of vertices in the $k$-neighbourhood of $v$. The $k$-neighbourhood can be thought of as a generalisation of the out-degree, except that we exclude a vertex from its own neighbourhood.

**Width-measures on Graphs.** Width-measures of graphs are based on cops-and-robbers games, where different measures are obtained by varying the rules of the game. Most of the measures have an alternative characterisation using specific sorts of graph decompositions. Cops and robber games were introduced by Nowakowski and Winkler [NW83] and, independently by Quilliot [Qui78], and have been well-studied, see *e.g.*, [ST93; Bar06; DKT97; GLS01; BG05; Hun07].

The classical width notion for *undirected graphs* is *treewidth*, which was originally defined by Robertson and Seymour [RS86], see also [Bod97]. Intuitively, the treewidth of a graph expresses how tree-like the graph is—the treewidth of a tree is 1. This corresponds to the idea that some problems are easier to solve for trees, or graphs that are almost trees, than for arbitrary graphs. For directed graphs, the treewidth is defined as the treewidth of the graph obtained by forgetting the direction of the edges. The complexity for solving parity games is bounded in the treewidth, as shown by Obdržálek [Obd03]; this means that, for parity games with a small, constant treewidth, parity game solving is polynomial.

It is not clear how treewidth should be generalised from undirected graphs to directed graphs. This question has led to a number of different width measures for directed graphs, of which we give an overview below.

*Directed treewidth* was introduced by Johnson *et al.* [Joh+01]. For a directed graph, its directed treewidth is bounded by its treewidth [Adl07].

*DAG-width* was proposed by Obdržálek [Obd03; Obd06b; Obd06a] and independently by Berwanger *et al.* [Ber+06], see also [Ber+12b]. This measure describes how much a graph is like a directed acyclic graph. It was shown that the DAG-width of a graph bounds the directed tree width of a graph from above, and that it is at most the tree-width of a graph.

*Kelly-width* was introduced by Hunter and Kreutzer [HK08]. It is known that if the Kelly-width of a graph is bounded, then also a bound on its directed tree-width can be given, however, classes of directed graphs with bounded directed treewidth and unbounded Kelly-width exist.

*Entanglement*, introduced by Berwanger and Grädel [BG05; Ber+12a], is a graph measure that aims to express how much the cycles in a graph are intertwined. If an undirected graph has bounded treewidth, say $k$, then it also has an entanglement bounded by $(k+1)\log|G|$. Furthermore, if a graph has DAG-width $k$, then it has entanglement at most $(k+1)\log|G|$.

*Clique-width*, introduced by Courcelle and Olariou [CO00], measures how close a graph is to a complete bipartite graph. For every directed graph with treewidth $k$, we know that its clique width is at most $2^{2k+1}+1$. Unlike the other width measures that we discussed, this measure does not have a characterisation in terms of cops-and-robbers games.

### 5.1.2 Parity Game Specific Properties

In addition to the above, we develop a number of additional characteristics of parity games, that use the player and priority information that is available. Henceforth, let $(V, \to, \Omega, \mathscr{P})$ be a parity game. We write $V_\Diamond$ (resp. $V_\Box$) as an abbreviation for $\{v \in V \mid \mathscr{P}(v) = \Diamond\}$ (resp. $\{v \in V \mid \mathscr{P}(v) = \Box\}$).

**Basic Sizes.** As basic parity game properties, we consider the numbers of vertices $|V_\Diamond|$ and $|V_\Box|$ owned by the two players. We write $\Omega(V)$ for the set of priorities $\{\Omega(v) \mid v \in V\}$, and represent the number of priorities in the game by $|\Omega(V)|$. Likewise, we represent the number of vertices with priority $k$ by $|\Omega^{-1}(k)|$.

**Local Structure.** For parity games, we characterise two more specific classes of diamonds. A diamond $(u, v, v', w)$ is defined to be *even* if $\mathscr{P}(u) = \mathscr{P}(v) = \mathscr{P}(v') = \Diamond$, and *odd* if $\mathscr{P}(u) = \mathscr{P}(v) = \mathscr{P}(v') = \Box$. These structures might prove interesting in the sense that from vertex $u$, $\mathscr{P}(u)$ has at least two strategies to play to $w$ in two steps.

**Alternation Depth.** Typically, the complexity of parity game algorithms is expressed in the number of vertices, the number of edges, and the number of priorities in the game. If we look at other verification problems, such as $\mu$-calculus model checking, or solving Boolean equation systems, the complexity is typically expressed in terms of the *alternation depth*. Intuitively, the alternation depth captures the number of alternations between different fixed point symbols. We first describe classical notions of alternation depth for $\mu$-calculus formulae. Subsequently we adapt this notion to parity games.

The original notion of alternation depth of a $\mu$-calculus formula was described by Emerson and Lei [EL86]. They used this notion to describe the complexity of their model checking algorithm. Some alternative definitions have also been described in the literature. Each of the definitions counts alternations in a subtly different way. The most popular alternatives are due to Niwinski [Niw86] and Anderson [And93].

The following overview of three notions of alternation depth is due to Bradfield and Stirling in [BS00]. This combines the notion of *simple-minded* alternation depth ($S$), Emerson-Lei alternation depth ($EL$), and Niwinski alternation depth ($N$). Bradfield described a slightly more involved definition of Emerson-Lei alternation depth in [Bra91]. The only difference between the three notions is clause 3 in the definition below.

**Definition 5.1.** [BS00] For $C \in \{S, EL, N\}$ We define classes $\sum_n^C$ and $\prod_n^C$ of least– and greatest fixed point expressions inductively as follows:

- $\phi \in \sum_0^C$ and $\phi \in \prod_0^C$ iff $\phi$ does not contain fixed point operators;
- $\sum_{n+1}^C$ is the smallest set such that, if $\phi \in \sum_n^C \cup \prod_n^C$ then $\phi \in \sum_{n+1}^C$, and furthermore
  1. if $\phi, \psi \in \sum_{n+1}^C$, then $\phi \vee \psi$, $\phi \wedge \psi$, $\neg\phi$, $\langle a \rangle \phi$, $[a]\phi \in \sum_{n+1}^C$;
  2. if $\phi \in \sum_{n+1}^C$ then $\mu Z.\phi \in \sum_{n+1}^C$;
  3. – if $C = EL$, we have that if $\phi(Z), \psi \in \sum_{n+1}^{EL}$, and $\psi$ is a closed formula, then $\phi(\psi) \in \sum_{n+1}^{EL}$.
     – if $C = N$ we have that if $\phi(Z), \psi \in \sum_{n+1}^N$, and no free variable of $\psi$ is captured by a fixed point operator in $\phi$, then $\phi(\psi) \in \sum_{n+1}^N$

$\prod_{n+1}^C$ is defined symmetrically, using $\nu$ instead of $\mu$ in the second clause above.

These classes give rise to the following three notions of alternation depth:

- simple minded alternation depth $\mathsf{ad}^{\mathsf{S}}(\phi)$ is the least $n$ such that $\phi \in \sum_{n+1}^{S} \cap \prod_{n+1}^{S}$,
- Emerson-Lei alternation depth $\mathsf{ad}^{\mathsf{EL}}(\phi)$ is the least $n$ such that $\phi \in \sum_{n+1}^{EL} \cap \prod_{n+1}^{EL}$, and
- Niwinski alternation depth $\mathsf{ad}^{\mathsf{N}}(\phi)$ is the least $n$ such that $\phi \in \sum_{n+1}^{N} \cap \prod_{n+1}^{N}$.

Observe that simple-minded alternation depth simply counts the alternations between fixed point symbols in a formula. Emerson-Lei alternation depth ignores closed subformulae, *i.e.*, it only counts alternations if one of the surrounding bound variables occurs within a subformula. Niwinski alternation depth is even more restrictive.

In coining a notion of alternation depth for parity games, we draw inspiration form alternation depth for modal equation systems as it was defined by Cleaveland *et al.* [CKS93].

The notion of alternation depth that we define comes in two stages. First we define the nesting depth of a strongly connected component within a parity game, next we define the alternation depth of the parity game as the maximum of the nesting depths of its strongly connected components.

**Definition 5.2.** Let $G = (V, \to, \Omega, \mathscr{P})$ be a parity game, and let $\mathscr{C}$ be the set of strongly connected components of $G$. Let $C \in \mathscr{C}$ be a strongly connected component. The nesting depth of $v_i$ in $C$ is given by

$$\begin{aligned}\mathsf{nd}(v_i, C) \stackrel{\Delta}{=} \max\{1,\\ &\max\{\mathsf{nd}(v_j, C) \mid v_j \to^*_{C,\Omega(v_i)} v_i, v_j \neq v_i \text{ and } \Omega(v_i) \equiv_2 \Omega(v_j)\},\\ &\max\{\mathsf{nd}(v_j, C) + 1 \mid v_j \to^*_{C,\Omega(v_i)} v_i \text{ and } \Omega(v_i) \not\equiv_2 \Omega(v_j)\}\\ \}\end{aligned}$$

where $v_j \to_{C,k} v_i$ if $v_j \to v_i$ is an edge in the SCC $C$ with $\Omega(v_j) \leq k$ and $\Omega(v_i) \leq k$. Intuitively, the nesting depth of a vertex $v$ counts the number of alternations between even and odd priorities on paths of descending priorities in the SCC of $v$. Note that this is well-defined since we forbid paths between identical nodes. The nesting depth of an SCC $C \in \mathscr{C}$ is defined as the maximum nesting depth of any vertices in $C$, *i.e.*, $\mathsf{nd}(C) \stackrel{\Delta}{=} \max\{\mathsf{nd}(v, C) \mid v \in C\}$.

The *alternation depth* of a parity game is defined as the maximal nesting depth of its SCCs.

**Definition 5.3.** Let $G = (V, \to, \Omega, \mathscr{P})$ be a parity game, and let $\mathscr{C}$ be the set of strongly connected components of $G$. Then the *alternation depth* of $G$ is defined as $\mathsf{ad}(G) \stackrel{\Delta}{=} \max\{\mathsf{nd}(C) \mid C \in \mathscr{C}\}$.

Cleaveland *et al.* showed that their notion of alternation depth for modal equation systems corresponds to Emerson-Lei alternation depth for formulae [CKS93, Theorem 3.8].

In Section 2.6 we already observed that for normalisation we have different possibilities. For the following proposition, consider a normalisation routine where alternating conjunctions and disjunctions are removed by introducing an additional equation immediately after the original one, instead of at the end of the BES. For example, the equation system $\mathscr{E}_0(\sigma X = f \wedge g)\mathscr{E}_1$ where $g$ is disjunctive, is replaced by $\mathscr{E}_0(\sigma X = f \wedge X')(\sigma X' = g)\mathscr{E}_1$.

The following proposition follows immediately from the result in [CKS93], and the encoding of model checking problems to parity games via $\mathsf{E}^L(\_)$, and subsequent normalisation using the above procedure.

**Proposition 5.4.** *Let $\phi$ be a modal $\mu$ calculus formula with alternation depth $\mathsf{ad}^{\mathsf{EL}}(\phi) \geq 1$, and let $L$ be an arbitrary labelled transition system. Let $G$ be a parity game obtained from $\mathsf{E}^L(\phi)$ after normalisation. Then $\mathsf{ad}(G) \leq \mathsf{ad}^{\mathsf{EL}}(\phi)$.*

Observe that using the original encoding can both lead to both smaller alternation depths as well as larger alternation depths.

### 5.1.3 Discussion

The graph properties that we described in Section 5.1.1 provide some basic characteristics of graphs. A high maximal degree, in a graph with a low average degree, *e.g.*, characterises that the graph contains a number of hubs that are incident to a lot of other vertices.

Algorithms and heuristics can benefit from a decomposition into strongly connected components (SCCs). One prominent example is the global parity game solving algorithm presented by Friedmann and Lange [FL09], for which it was shown that SCC decomposition generally works well in practice. Furthermore, Berwanger and Grädel showed that, for certain subclasses of parity games, all vertices in a strongly connected component are included in the same winning set [BG04].

Graph algorithms are typically based on a search strategy such as breadth-first search (BFS) or depth-first search (DFS), given some initial vertex $v_0 \in V$. This is the main reason to study characteristics for these measures.

The diameter and the girth characterise global properties of graphs. Intuitively, they describe how hard it gets to get from one vertex in the graph to another, or back to itself. A girth of 1 denotes that the graph contains a self-loop. We expect to see this value quite often when analysing parity games due to the occurrence of vertices that are trivially won by one of the two players.

Even- and odd diamonds could be interesting in parity games. Consider, *e.g.*, an even diamond $(u, v, v', w)$. This means that from vertex $u$, player $\Diamond$ has two strategies to reach $w$. The question is open whether these kinds of structures can be used to improve parity game solving.

Width-measures of graphs are interesting to consider for parity games due to the availability of specialised algorithms that can solve games polynomially if their width is bounded.

Our notion of alternation depth is an attempt to devise a complexity measure for parity games that is more accurate than the number of priorities.

Most of the measures that we have described above can be computed efficiently. However, the diameter and girth are quadratic in the number of vertices—effectively they require computing a solution to the all-sources shortest path problem. This is not feasible for even relatively small graphs.

The situation for the width-measures is even worse. Computing the exact value for these measures is known to be NP-complete [ACP87]. Approximation algorithms are known that compute upper- and lower bound for these measures; especially for treewidth these have been thoroughly studied [BK10; BK11]. However, even these algorithms are not practical for the sizes of graphs that we consider, for these algorithms, even graphs with only 1000 vertices are considered large. Due to the availability of specialised algorithms, we would have liked to include the width-measures in the statistics of our benchmarks. Regretfully, due to the poor running times, including this information is not feasible.

## 5.2 Parity Games

For benchmarking parity game algorithms, it makes sense to distinguish three classes of parity games, (1) the games that are the result of encoding a problem into parity games, (2) games that represent hard cases for certain algorithms, and (3) random games. All three classes of games occur in the literature, and our benchmark set contains games from each of these classes. In the rest of this section we discuss the set of benchmarks that we have used.

### 5.2.1 Encodings

A broad range of verification problems can be encoded as a parity game. The most prominent examples of these are the $\mu$-calculus model checking problem—does a model satisfy a given property?—, equivalence checking problems—are two models equivalent?—, decision procedures—is a formula valid or satisfiable?— and synthesis—given a property, give a model that satisfies the property.

#### Model Checking

The set of model checking problems we consider is mainly selected from the literature. All of the systems are encodings that, given a model $L$ of a system, and a property $\varphi$, encode the model checking problem $L \models \varphi$, *i.e.*, does $L$ satisfy property $\varphi$. Most sensible encodings of model checking problems typically lead to a low number of priorities, corresponding to the low alternation depths of these properties. We verify fairness, liveness and safety properties. This set includes, but is not limited to, the model checking problems described in [Mat03; PW08; FL09; CKW11; CKW12b].

We considered a number of communication protocols from the literature, see, *e.g.*, [BSW69; CK74; KM90; GP96]: two variations of the *Alternating Bit Protocol* (ABP), the *Concurrent Alternating Bit Protocol* (CABP), the *Positive Acknowledgement with Retransmission Protocol* (PAR), the *Bounded Retransmission Protocol* (BRP), the *Onebit* sliding window protocol, and the *Sliding Window Protocol* (SWP). All of these protocols are parameterised with the number of messages that can be sent, furthermore the sliding window

protocol is parameterised by the window size. For these protocols a number of properties of varying complexity was considered, ranging from alternation free properties, *e.g.* deadlock freedom, to fairness properties.

A *Cache Coherence Protocol* (CCP) [Vel+01] and a *wait-free handshake register* (Hesselink) [Hes98] are considered. For the cache coherence protocol we consider a number of properties from [Pan+07] and for the register we consider properties from [Hes98]. Additionally we consider a *leader election protocol* for which we verify whether it eventually stabilises.

To obtain parity games with a high degree of alternation between vertices owned by different players we also consider a number of two-player board games, *viz. Clobber* [Alb+05], *Domineering* [Gar74], *Hex*, see *e.g.* [BBC00; Maa05], *Othello*, also known as reversi, see *e.g.* [Ros05], and *Snake*. For these games we check for each of the players whether the player has a winning strategy starting from the initial configuration of the game. The games are parameterised by their board size.

Additionally, we consider a number of industrial model checking problems. The first is a system for lifting trucks (Lift) [GPW03], of which we consider both a correct and an incorrect version. We verify the liveness and safety properties described in [GPW03]. For the *IEEE 1394 Link Layer Protocol* (1394) we verify the properties from [Lut97]. We translated the ACTL properties from [SM98] to the $\mu$-calculus.

Finally, we check the *Elevator* described by Friedmann and Lange, in a version in which requests are treated on a first-in-first-out basis (FIFO), and on a last-in-first-out basis (LIFO). We then check whether, globally, if the lift is requested on the top floor, then it is eventually served. This holds for the FIFO version, but does not hold for the LIFO version of the model. The elevator model is parameterised by the strategy and the number of floors. Furthermore we consider the parity games generated using an encoding of an LTS with a $\mu$-calculus formula, as well as the direct encoding presented in [FL09]. In a similar way we consider the Hanoi towers from [FL09] as well as our own version of this problem. An overview of the $\mu$-calculus formulae that we considered is provided in Appendix A.1.

#### Equivalence Checking

Given two processes $L_1, L_2$, the problem whether $L_1 \equiv L_2$, for relations $\equiv$, denoting that $L_1$ and $L_2$ are equivalent under some process equivalence, can be encoded as a parity game [Lar93; VL94; Mat03; Che+07]. We consider strong bisimulation, weak bisimulation, branching bisimulation and branching simulation equivalence in our benchmarks, using the approach described in [Che+07]. The number of different priorities in these parity games is limited to 2. The games do, however, include alternations between vertices owned by different players.

Here we again use the specifications of the communication protocols that we also used for model checking, *i.e.*, two ABP versions, CABP, PAR, Onebit and SWP. In addition we include a model of a buffer. We vary the capacity of the buffer, and the number of messages that can be transmitted, as well as the window size in the sliding window protocol. We compare each pair of protocols using all four equivalences. This gives rise to both positive and negative cases. These cases are a superset of the ones described in [CKW11; CKW12b].

In addition, we include a comparison of the implementation of the wait-free handshake register with a possible specification. The implementation is trace equivalent to the specification, but it is not equivalent with respect to the equivalences that we consider here.

#### Decision Procedures

Parity games can also be obtained from decision procedures for temporal logics such as LTL, CTL, CTL$^*$, PDL and the $\mu$-calculus. Classical approaches rely on testing non-emptiness of a tree automaton, see, *e.g.*, [EJ99]. Friedmann, Latte and Lange presented a decision procedure that is based on a combination of infinite tableaux in which the existence of a tableau is coded as a parity game [FLL10]. The priorities in the parity game originate from a deterministic parity automaton that is the result of complementing a non-deterministic Büchi automaton. In *ibid.*, the authors also present a tool that, for a given formula, checks whether it is (1) *valid*, *i.e.*, whether the formula holds in all models, or (2) *satisfiable*, *i.e.*, whether the formula is satisfiable in some model.

Our benchmark set includes a number of scalable satisfiability and validity problems that are provided as examples for the MLSolver tool [FL10a], including, but not limited to, the benchmarks used in [FL10a]. The formulae are provided in Appendix A.2.

#### Synthesis

Another problem that involves solving parity games is the LTL synthesis problem. Traditional synthesis approaches convert a formula into a non-deterministic Büchi automaton, which is, in turn, transformed into a deterministic parity automaton using Safra's construction [Saf88]. Emptiness of this deterministic parity automaton can then be checked using parity games. Implementations of this approach suffer from the high cost of determinisation, even for small automata.

Synthesis tools have been implemented that employ parity games internally. GOAL [Tsa+08], *e.g.*, converts an LTL formula to an equivalent deterministic parity automaton. This can be converted into a parity game using, *e.g.*, Gist [Cha+10]. All synthesis tools that we are aware of, however, are research quality tools, of which we have not been able to obtain working versions on current computing platforms. As a result, we do not currently include parity games obtained from the synthesis problem.

### 5.2.2 Hard Games

The interesting complexity of solving parity games, and its link to the model checking problem, have led to the conception of a large number of parity game solving algorithms. For most of these algorithms it has long been an open problem whether they have exponential lower bounds.

We consider the games described by Jurdziński that shows the exponential lower bound for small progress measures [Jur00], the *ladder games* described by Friedmann [Fri11a] with the variation of *recursive ladder games* that give a lower bound for the strategy improvement algorithms [VJ00; Sch07], as well as *model checker ladder games*

for which the local algorithm by Stevens and Stirling [SS98] requires exponential time as described in [Fri10].

In our benchmarks, we include these families of games.

### 5.2.3 Random Games

The final class of games that is typically used in publications that empirically evaluate the performance of algorithms on parity games are random parity games. In the literature [BV01; Sch08b; Sch08a; Lan05; FL09], random games we discussed. We study three classes of random parity games. We expect that the structural properties of random games are, typically, different from parity games obtained in the previous classes. This class is, therefore, unlikely to give insights in the performance of parity game algorithms on practical problems.

## 5.3 Implementation

All experiments were executed on a 1TB main memory, 56-core Linux machine, where each core was running at 2.27GHz. Executions of tools generating, reducing and solving parity games, as well as executions of tools collecting statistics about parity games, were limited to running times of 1 hour and their memory usage was limited to 32GB.

To systematically generate the benchmarks, and perform experiments, we have implemented tooling to drive the experiments that allows the parallel execution of individual cases. Here a case is either generating, solving or reducing a game, or collecting a single measure. Each individual case only uses a single core. The tools are available from https://github.com/jkeiren/paritygame-generator. Care was taken to implement the tooling in an extensible way, *i.e.*, additional parity games, additional encodings, as well as additional measures can be added straightforwardly.

### 5.3.1 Generating Parity Games

For the generation of our benchmarks we rely on a number of external tools. We used version 3.3 of PGSolver [FL10b] for generating random games, and games that prove to be hard for certain algorithms. Version 1.2 of MLSolver was used to generate the games for satisfiability and validity problems [FL10a]. For the model checking and equivalence checking problems we used revision 11703 of the mCRL2 toolset [Cra+13]. This toolset is based on (parameterised) Boolean equation systems, and hence requires a conversion to parity games. Results can differ for different translations to simple recursive form, that are used in this conversion. The implementation uses the transformation described in Section 2.5.

We have applied the reductions from Chapter 4 to each of the parity games generated using the tools described above. For all games we have collected the information described in Section 5.1.

### 5.3.2 Collecting Statistics

We developed the tool `pginfo` for collecting structural information from parity games. The tool is available from https://github.com/jkeiren/pginfo and accepts parity games in the file format used by PGSolver. It reads a parity game, and writes statistics out to a file in a structured way.

The implementation is built on top of the Boost Graph library [SLL02], which provides data structures and basic algorithms for manipulating graphs. Most of the measures can be effectively computed using the BFS and DFS algorithms implemented in the library. To determine feasibility of computing width-measures for our benchmarks we have implemented three approximation algorithms. For computing upper and lower bounds on treewidth we implemented the greedy degree algorithm [BK10] and the minor min-width algorithm [GD04], respectively. For computing an upper bound of the Kelly-width we implemented the elimination ordering described in [HK08]. These approximation algorithms have proven to be impractical due to their complexity. Computing (bounds) on the other width measures is equally complex.

### 5.3.3 Equivalence Reductions

The equivalence reductions described in Chapter 4 have been implemented in the tool `pgconvert`, which is available from https://github.com/tue-mdse/pgconvert. This tool implements the algorithm described in the previous chapter for deciding governed stuttering bisimilarity. For stuttering equivalence the classical algorithm by Groote and Vaandrager was implemented. The algorithms for strong bisimulation and governed bisimulation are also simplifications of the same algorithm, as a consequence the running time is not the $\mathcal{O}(n \log n)$ that can be achieved theoretically, but instead it is an $\mathcal{O}(nm)$. However, since all algorithms were implemented in the same framework, we are able to compare the effect of the specific equivalences. Note that for strong bisimulation and governed bisimulation a more efficient implementation could be made on top of the partition refinement algorithm by Paige and Tarjan [PT87].

We have validated the implementation using the translation from [RSW12], using existing tools for strong- and divergence-preserving branching bisimulation for reducing labelled transition systems, for a subset of the parity games.

## 5.4 Analysis of Benchmarks

We have presented a diverse set of benchmarks by considering games originating from different problems. Next we analyse our benchmarks with respect to the measures described in Section 5.1. This way we illustrate that the benchmarks we have chosen contain games with a wide variety of properties. Furthermore, this gives us some insights in the characteristics of typical parity games. Note that for each of the statistics that we present, we consider only the parity games for which that specific statistic could be computed within an hour. We used this bound to avoid timeouts for computing the measures that are expensive to compute, such as the diameter and the girth. All graphs in this section, as well as the next, are labelled by their class. Note that the satisfiability and

validity problems are labelled by “mlsolver” and the games that are hard for some solving algorithms are labelled by “specialcases”. The full data presented in this chapter is available as [Kei13].

We have considered 951 parity games that range from 2 vertices to 4.9 million vertices, and on average they have about 90,000 vertices. The number of edges ranges from 2 to 100 million, with an average of about 465,000. The games are a mixture of parity games in which all vertices are owned by a single player, the so called solitaire games [BG04], and parity games in which both players own non-empty sets of vertices. The parity games that we consider have differing degrees. There are instances in which the average degree is 1, the average degree is maximally 9999, but it is typically below 10. The ratio between the number of vertices and the number of edges is, therefore, relatively small in general. This can also be observed from Figure 5.1, which displays the correlation between the two. The games in which these numbers coincide are on the line $x = y$, the other games lie around this line due to the log scale that we use. Our parity games generally contain a vertex with in-degree 0, which is the starting vertex. Typically the games contain vertices with a high in-degree—typically representing vertices that are trivially won by either of the players—, and vertices with a high out-degree.

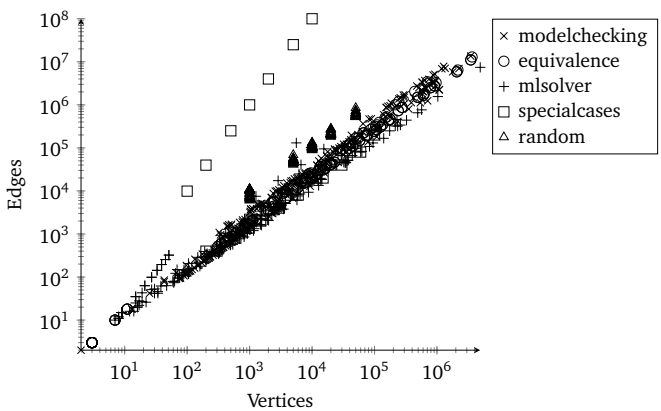



Figure 5.1: Relation between number of vertices and number of edges

In general, the SCC quotient height ranges up to 513 for the parity games that we consider. The number of non-trivial SCCs, can grow large, up to 125,000 for our games.

The diameter and girth have been computed only for smaller parity games, and the data we present for them, therefore, considers a subset of the parity games only.

We expect that typical parity games contain self-loops, which leads to a small girth—the girth is 1 if the game contains a self-loop. This is confirmed by the data in Figure 5.2. Note that the girth is large for some of the hard cases that we consider. A closer investigation shows that this is solely due to the model checker ladder games [Fri10].

The diameters of the parity games, *i.e.*, the maximal length of any shortest path in the game, are nicely distributed over the sizes of the game. Figure 5.3 shows that for every size of game we have parity games of a large range of different diameters. Observe that for the hard cases the diameter is, generally, large. Again, this is due to variations of ladder games. Generally, the diameter for satisfiability and validity problems is larger

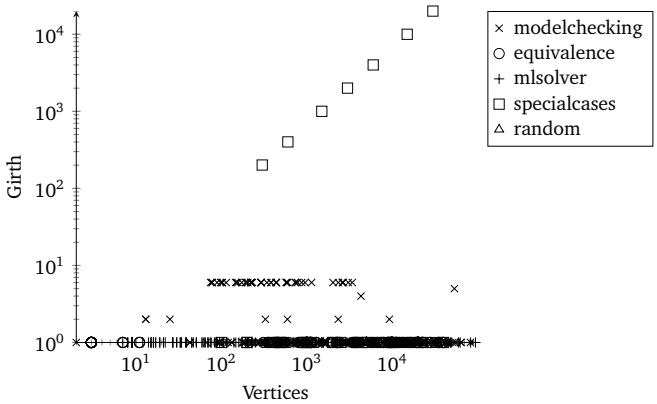



Figure 5.2: Girths

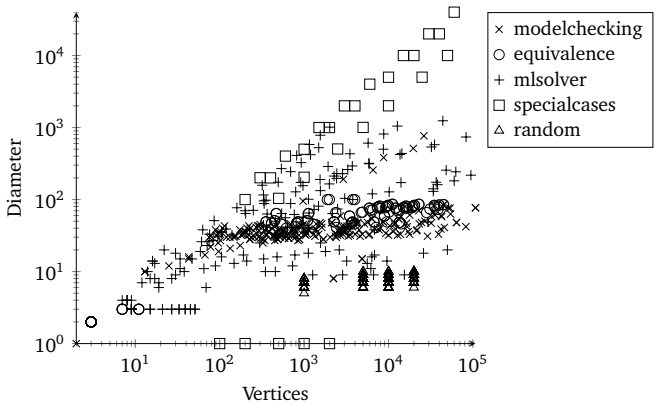



Figure 5.3: Diameters. Note the logarithmic scale for the diameter

than the diameter for model checking problems. For random games the diameter is typically small.

Figure 5.4 indicates that the diameter and the number of BFS levels are correlated, the number of BFS levels is therefore likely to be a good approximation of the diameter, also for larger instances. Note that this corresponds to a similar observation made by Pélanek, who stated that typically the diameter is smaller than 1.5 times the number of BFS levels for state spaces [Pel04].

Of the parity games that we consider, 725 contain diamonds. Of these, 483 contain even diamonds, and 624 contain odd diamonds, 382 contain both. This indicates that it is worth investigating techniques, such as confluence reduction, that use these diamonds to either simplify parity games or speed up solving. In general, the number of diamonds is independent of the number of vertices in the game.

The 3-neighbourhoods range from 3 to 3000 across the sizes of the games, as can be seen from Figure 5.5. Note that the average size of the 3-neighbourhoods is typically

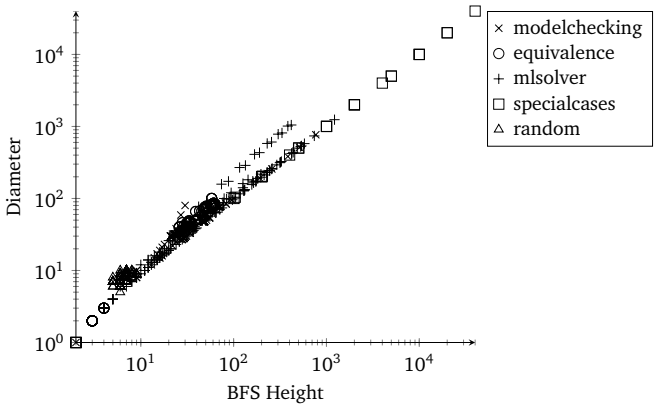



Figure 5.4: Correlation between BFS levels and diameter

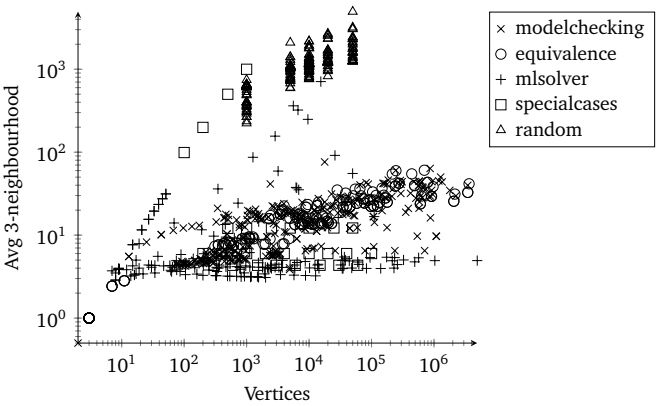



Figure 5.5: 3 Neighbourhoods

high for random games, and limited to 100 for most other classes of games.

We have included parity games with alternation depths up to 10,000 as shown in Figure 5.6. Observe that the games for model checking and equivalence checking all have alternation depth at most 2—which is lower than alternation depth 3 of some of the formulae due to the normalisation procedure that we have used. For model checking properties could be formulated that have a higher alternation depth—up to arbitrary numbers—however, in practice properties have limited alternation depth because they become too hard to understand otherwise. The satisfiability and validity properties have alternation depths between 1 and 4. The alternation depths of the random games are between 10 and 15. All parity games with more than 50 priorities represent special cases. Closer investigation shows that these special cases are the clique games and recursive ladder games.

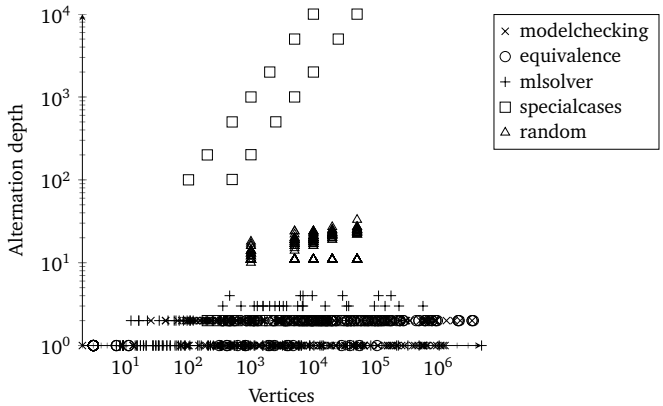



Figure 5.6: Alternation depths

## 5.5 Evaluation of Equivalence Reductions

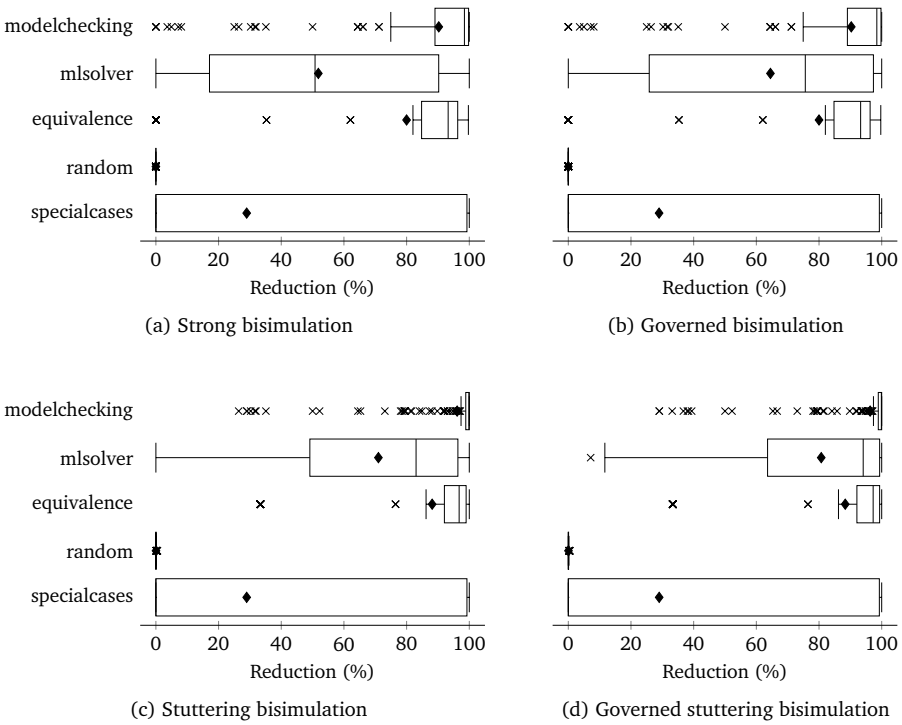
Now that we have established a representative set of parity games, we can evaluate the effect of the equivalence reductions developed in the previous chapter. We first look at the sizes of the reduced parity games, and then study the impact on the time required for solving.

In Figure 5.7 we present an overview of the size reduction for each of our four equivalences as a box plot. The boxes indicate the median with the lower- and upper quartiles, such that a quarter of the measurements is between the lower quartile and the median, and another quarter is between the median and the upper quartile. The lower- and upper whiskers are determined by subtracting (resp. adding) 1.5 times the distance between the upper and lower quartile. The average reductions are shown as solid diamonds, outliers are marked as $\times$.

Reductions are presented as percentages, where *e.g.*, a reduction of 90% means that the size of the resulting system is 10% of the original. Note that the size considered is the sum of the number of vertices and the number of edges in the parity game.

The results show that all of the equivalences are able to reduce the size of parity games obtained from model checking problems by about 90% on average. For strong bisimulation and governed bisimulation there are cases in which there is no reduction at all, and for all equivalences there are instances where the reduction effectively solves the parity game, *i.e.*, the reduction reduces the game to a single vertex with a self-loop, shown as (approximately) 100% in the figures. For model checking cases, observe that stuttering equivalence and governed stuttering equivalence always achieve a reduction of at least 25% for the cases that we considered; the average reduction has increased to about 95%, and the median approaches 100%.

For satisfiability and validity examples the average reduction using strong bisimulation is only 50%. This reduction is improved to 65 to 70% by governed bisimulation and stuttering bisimulation, and up to 80% by governed stuttering bisimulation. This is expected due to the alternations between vertices owned by different players in this type of parity game.




(a) Strong bisimulation

(b) Governed bisimulation

(c) Stuttering bisimulation

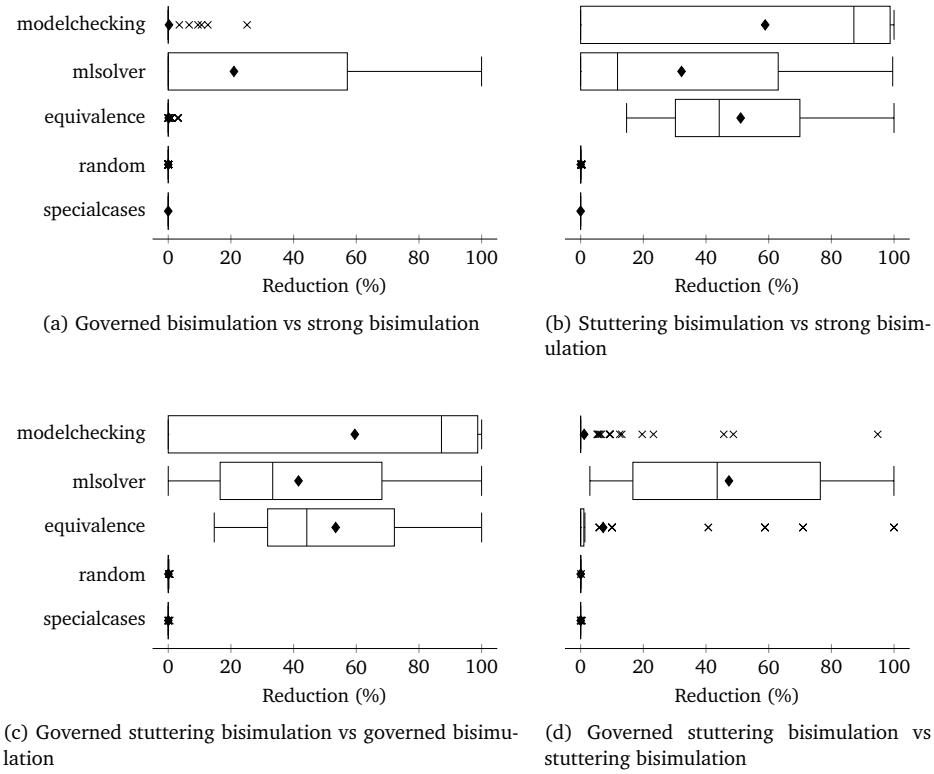
(d) Governed stuttering bisimulation

Figure 5.7: Reduction percentages compared to original.

The parity games that were obtained for equivalence checking show large reductions using all algorithms, with medians of over 90%, and averages ranging from roughly 75% for strong bisimulation, to 85% for governed stuttering equivalence. Random games can, in general, not be reduced using our reduction techniques, which is shown by the narrow range of values in the figures for random games. This should not come as a surprise, since equivalence reductions rely on regularities in the structure of parity games. This regularity is absent in random games.

The games representing special cases can, on average, be reduced by about 25%. The ladder games [Fri11a] are reduced to 2 vertices, with 2 edges by our reductions. Note that the model checker ladder games are not reduced by any of our reductions, and from the recursive ladder games governed stuttering is able to remove only 3 vertices, whereas the other equivalences do not result in any reduction.

From the graphs in Figure 5.7 the relative reductions of the different equivalences are hard to judge. We therefore present the relative reductions in Figure 5.8. Observe that governed bisimulation does not prove to be an improvement beyond strong bisimulation in most cases, confirming the results from [KW11]. As expected, the most significant effect is observed in the satisfiability and validity cases, where we know there are alternations between vertices owned by different players. Stuttering bisimulation is




(a) Governed bisimulation vs strong bisimulation

(b) Stuttering bisimulation vs strong bisimulation

(c) Governed stuttering bisimulation vs governed bisimulation

(d) Governed stuttering bisimulation vs stuttering bisimulation

Figure 5.8: Relative reductions.

an improvement over strong bisimulation; on average it is able to reduce the games by an additional 50%. For model checking, the median is around 85%. The improvement of governed stuttering equivalence over governed bisimulation is similar. For the problem sets in which there are a lot of alternations between vertices owned by different players, *i.e.*, equivalence checking and satisfiability and validity checking, we see that governed stuttering bisimulation still improves over stuttering equivalence, sometimes up to 100%. Note that there are examples in which governed stuttering equivalences is unable to achieve a reduction, most likely due to alternations between vertices with different priorities. The reduction using governed stuttering equivalence for the model checking and equivalence checking cases might be improved by altering the transformation to SRF, which is used in mCRL2, to add newly added equations to the block from which they were created, instead of to the end of the equation system.

In Table 5.1 we present a representative selection of our benchmarks that shows the absolute sizes of the corresponding parity games.

So far we have seen that the equivalences are able to dramatically reduce the size of parity games. It remains to be seen, however, whether first reducing the parity game, and only then solving it, also speeds up parity game solving. In Figure 5.9 we show the

Table 5.1: Absolute sizes of a selection of parity games. The numbers reported reflect the number of vertices in the parity games. Verdict $\surd$ means the outcome of the verification problem is true; $\times$ means the outcome is false. The buffers are of capacity 1.

| | | Original | $\leftrightarrows$ | $\rightleftarrows$ | $\simeq$ | $\backsimeq$ | verdict |
|---|---|---|---|---|---|---|---|
| Model Checking Problems | | | | | | | |
| **No deadlock** | | | | | | | |
| *Onebit* | $\|D\| = 2$ | 81 921 | 1 | 1 | 1 | 1 | $\surd$ |
| | $\|D\| = 3$ | 289 297 | 1 | 1 | 1 | 1 | $\surd$ |
| **No spontaneous generation of messages** | | | | | | | |
| *Onebit* | $\|D\| = 2$ | 185 089 | 1 | 1 | 1 | 1 | $\surd$ |
| | $\|D\| = 3$ | 1 278 433 | 1 | 1 | 1 | 1 | $\surd$ |
| | $\|D\| = 4$ | 5 588 481 | 1 | 1 | 1 | 1 | $\surd$ |
| **Messages that are read are inevitably sent** | | | | | | | |
| *Onebit* | $\|D\| = 2$ | 153 985 | 1 746 | 1 746 | 2 | 2 | $\times$ |
| | $\|D\| = 3$ | 579 745 | 1 746 | 1 746 | 2 | 2 | $\times$ |
| **Black has a winning strategy** | | | | | | | |
| *Snake* | $4 \times 4$ | 4 860 | 655 | 482 | 465 | 230 | $\times$ |
| *Clobber* | $4 \times 4$ | 564 914 | 29 200 | 26 725 | 28 357 | 23 796 | $\surd$ |
| **Values written to the register can be read** | | | | | | | |
| *Hesselink* | $\|D\| = 2$ | 1 093 761 | 1 | 1 | 1 | 1 | $\surd$ |
| Equivalence Checking Problems | | | | | | | |
| **Branching bisimulation equivalence** | | | | | | | |
| *ABP-CABP* | $\|D\| = 2$ | 57 905 | 6 143 | 6 143 | 4 124 | 4 070 | $\surd$ |
| | $\|D\| = 4$ | 134 097 | 6 143 | 6 143 | 4 124 | 4 070 | $\surd$ |
| *Buf(1)-Onebit* | $\|D\| = 2$ | 604 354 | 12 826 | 12 826 | 9 258 | 9 258 | $\times$ |
| **Weak bisimulation equivalence** | | | | | | | |
| *ABP-CABP* | $\|D\| = 2$ | 50 865 | 2 286 | 2 286 | 846 | 846 | $\surd$ |
| | $\|D\| = 4$ | 118 225 | 2 286 | 2 286 | 846 | 846 | $\surd$ |
| *Buf(1)-Onebit* | $\|D\| = 2$ | 631 474 | 6 657 | 6 657 | 3 140 | 3 140 | $\times$ |
| Satisfiability and validity checking problems | | | | | | | |
| **Limit closure** | | | | | | | |
| CTL | $n = 4$ | 38 075 | 3 679 | 3 665 | 1 083 | 306 | $\surd$ |
| | $n = 5$ | 60 011 | 8 223 | 8 202 | 2 888 | 916 | $\surd$ |
| CTL$^*$ | $n = 1$ | 174 667 | 80 656 | 80 632 | 26 876 | 12 471 | $\times$ |
| **Binary counter** | | | | | | | |
| PDL | $n = 7$ | 1 031 612 | 9 928 | 9 925 | 3 164 | 3 | $\times$ |
| | $n = 8$ | 4 924 413 | 21 961 | 21 958 | 7 250 | 3 | $\times$ |
| **Parity vs. Büchi condition** | | | | | | | |
| *Original* | $n = 2$ | 2 497 | 1 238 | 1 236 | 710 | 541 | $\surd$ |
| | $n = 3$ | 33 969 | 14 311 | 14 252 | 5 601 | 4 520 | $\surd$ |
| *Compacted* | $n = 2$ | 456 | 207 | 207 | 207 | 155 | $\surd$ |
| | $n = 3$ | 6 182 | 1 683 | 1 683 | 1 683 | 1 281 | $\surd$ |

solving times of the original, compared to the sum of the reduction and solving times of the reduced games modulo governed bisimulation. The comparison with the other equivalences is similar. We have used a timeout of an hour for reducing + solving, and

timeouts are shown in the graphs as a value of an hour, *i.e.*, 3,600 seconds.

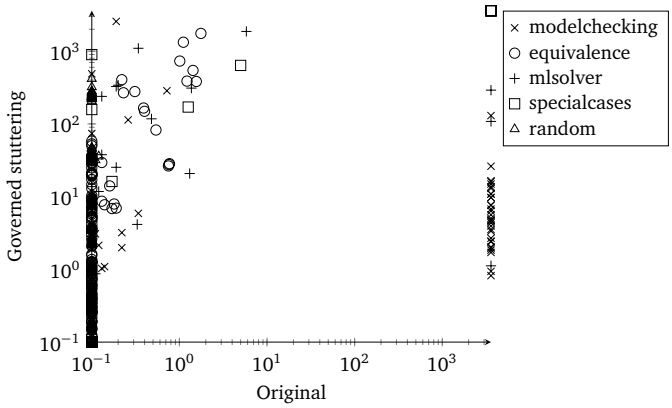


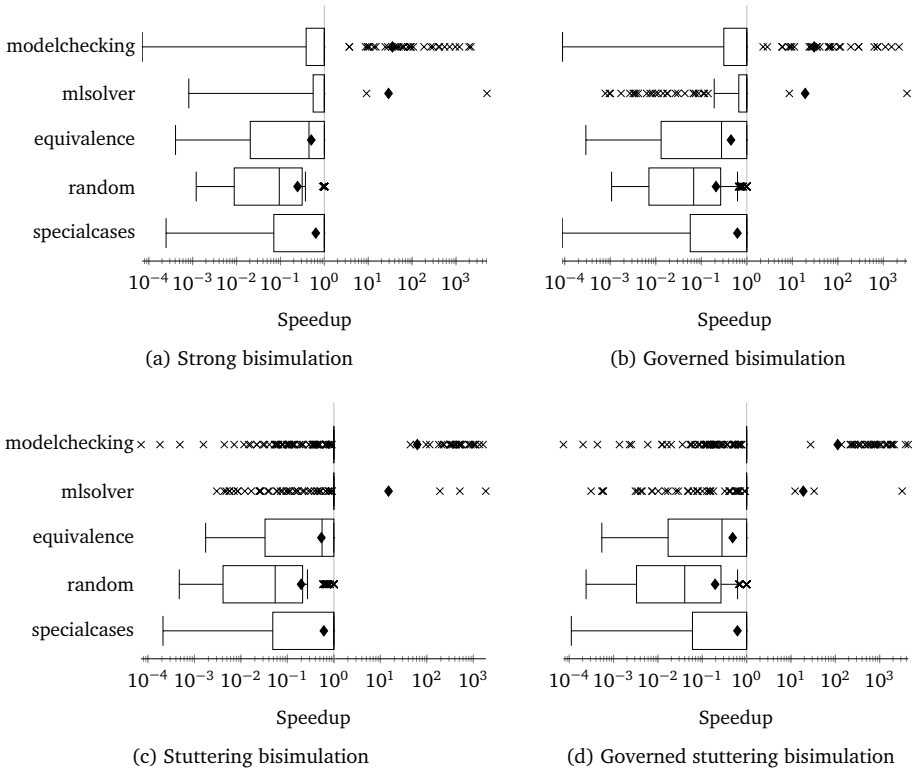

Figure 5.9: Solving (and reduction) times of the original games compared to governed stuttering bisimulation reduced games.

Points below the line $x = y$ indicate games in which first reducing and then solving the games is beneficial, whereas points above this line indicate that directly solving the original game is faster. In this graph, we can observe that a large number of parity games has been analysed, and that the majority of times is above the line $x = y$. For other equivalences the graphs are similar, and therefore omitted.

Figure 5.9 suggests that, in general, it is not wise to reduce a parity game. To further study this observation, we provide box plots showing the speed-up of reducing and solving compared to solving the original game in Figure 5.10. The speed-up is computed as time for solving original/(time for reduction + time for solving reduced game), the vertical line in plots indicates a speed-up of 1. The speed-up corresponding to the results in Figure 5.9 can be found in Figure 5.10d.

The results again indicate that, in general, solving the original game is faster than first reducing a game, and subsequently solving the reduced game. For model checking and satisfiability and validity problems, and the games that are hard for certain algorithms, the median speed-up is around 1 for all reductions that we have investigated. For the other problems the median indicates a slow-down. Note that the range of speed-up values is large. From Figure 5.9 we learn that in a large number of cases, both running times are below 5 seconds. Inaccuracy of measurements could lead to extremely high or extremely low speed-up values for these small running times. To investigate whether the speed-up changes if these low running times are ignored, we show those cases in which at least one of the two approaches takes 5 seconds in Figure 5.11. Here we see that for the model checking problems, the median value shows a speed-up in favour of governed stuttering equivalence reduction, over applying no reduction at all, whereas for the other problems applying governed stuttering equivalence reduction causes a slow down, with a small distribution of the values.

Our observations differ from the observations made earlier in [CKW11; CKW12b], in which it was shown that first reducing and then solving the game was competitive with directly solving the game, *i.e.*, most of the points were around the diagonal in plots




(a) Strong bisimulation

(b) Governed bisimulation

(c) Stuttering bisimulation

(d) Governed stuttering bisimulation

Figure 5.10: Speed-up with respect to original. Note the log-scale of the $x$-axis.

similar to Figure 5.9. If we translate these observations to the speed-ups, we observe that the range of speed-ups was much tighter in these earlier experiments. The observation that, using reduction, we are sometimes able to solve games that could not be solved efficiently with the tested implementations is, however, confirmed by our results.

The main difference between the experiments carried out in this chapter and those in [CKW11] and [CKW12b], is that here we have used a C++ implementation of the recursive algorithm in the tool `pbespgsolve`, which was not yet available at the time of writing of the previous papers. Earlier results were, therefore, based on a C++ implementation of the *small progress measures* algorithm in `pbespgsolve`, as well as the implementations of both algorithms and the big step algorithm in PGSolver [FL10b].

The implementation of the recursive algorithm that we have used here proves to be much more efficient than the implementations that were used in earlier experiments. This also proves the warning that was made in [CKW12b] to be just, *i.e.*, the results from such comparisons can change dramatically when different implementations of the same algorithm are used.

In Chapter 4 we hypothesised that, although the worst-case running time for governed stuttering equivalence is worse than that of stuttering equivalence, the extra factor will generally not manifest itself in practice. To investigate this hypothesis we show the

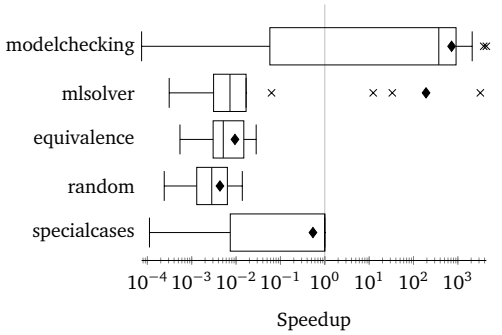



Figure 5.11: Speed-up of governed stuttering bisimulation with respect to original, for cases in which the running time of at least one of the two exceeds 5 seconds.

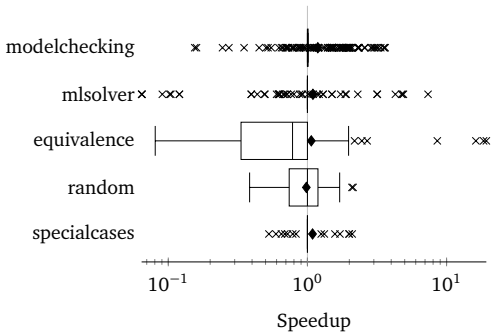



Figure 5.12: Speed-up of governed stuttering bisimulation with respect to stuttering equivalence.

speed-up of governed stuttering bisimilarity reduction + solving, with respect to stuttering equivalence reduction + solving, in Figure 5.12. The only case in which a small additional overhead is apparent, is in the parity games for equivalence checking problems. Observe, however, that in all other cases the mean speed-up is close to one, indicating no difference. Furthermore, the range of speed-up values is small, with some outliers. Based on these observations, we conclude that our hypothesis is supported by the data, *i.e.* the higher theoretical complexity of the algorithm for governed stuttering bisimilarity does generally not show in practice.

Finally, we have investigated whether any of the measures that we introduced has an effect on the reduction and solving times, *i.e.*, are the reduction and solving times correlated with one of the measures that we have described. No such correlation was observed, so none of the measures seems to have a clear impact on reduction and solving times.

## 5.6 Closing Remarks

In this chapter we have presented a set of benchmarks for parity game algorithms, and analysed the parity games based on their structural properties. These games were used to determine the reducing capabilities of the reductions presented in the previous chapter.

Our experiments show that the size reductions that can be obtained using the equivalences are significant, especially for practical problems like model checking, equivalence checking, as well as satisfiability and validity checking. Size reductions of all equivalences are significant, but governed stuttering equivalence is capable of achieving a significantly larger reduction of parity games, especially when alternation between vertices owned by different players is present.

The running times are, generally, negatively impacted by the reductions. There are, however, cases where directly solving the original game results in a timeout, whereas first reducing the game and subsequently solving it finishes quickly.

With the current solving techniques, in practice more time is spent generating the parity game for a verification problem, *e.g.* from a symbolic description, than is spent on solving the parity game. The generation time could potentially be reduced by reducing this symbolic description prior to generating the parity game. The large size reductions that we have shown in this chapter indicate that this is a viable alternative. We explore such symbolic reductions in the next chapter.

We described a set of measures for parity games, and studied their effect on the time required for reducing and solving the games. We were unable to establish a correlation between these measures and the required times. This leads to the question whether there are more suitable measures for parity games, for which such correlations can be established.

Finally, we proposed a notion of alternation depth for parity games. The alternation depth is always at most the number of priorities in a parity game. It is an open problem whether the complexity of any of the existing algorithms is such that it solves parity games exponentially in the alternation depth, instead of the number of priorities in the game.

# Chapter 6

# Liveness Analysis for Parameterised Boolean Equation Systems

So far we have studied equivalence relations for parity games and Boolean equation systems. In the previous chapter we have seen that these reductions have the potential to dramatically reduce the size of parity games. Looking at the time improvements, results are disappointing. Yet there are parity games that can be solved effectively with equivalence reductions, that cannot be solved without them. The main drawback of the approaches that we have considered thus far is that they still require the explicit construction of the complete parity game, prior to reduction. In this chapter, we focus on symbolic descriptions of parity games, and we describe a static analysis technique that is able to reduce the underlying parity game prior to generation.

It is well-known in explicit model checking that first generating the state space and reducing it a posteriori can be inefficient, since the state space is large, and its generation time consuming. A *static analysis* of a symbolic representation of a system may be able to reduce the state space size a priori [HM97; GL02]. For instance, in [PT09] the control flow in a system was used to analyse its data flow, leading to significant reductions compared to other known static analysis techniques.

In [FBG03], it has been suggested that by including the properties to be verified in the analysis, the effectiveness of the static analysis techniques can be improved. This is a challenging task, since both the property and the specification need to be analysed simultaneously.

A natural formalism for exploring such static analysis techniques are *Parameterised Boolean Equation Systems* (PBESs) [GW05b; GW05a]. These generalise Boolean equation systems and can be used for solving a variety of verification problems, such as the encoding of first order $\mu$-calculus model checking problems over (possibly infinite) labelled transition systems [GW05b; GW05a] and equivalence checking of various behavioural equivalences on labelled transition systems [Che+07]. For model checking, *e.g.*, the system and the property are translated into a PBES, which hence includes the information of

both the system and the property that is being verified. As a consequence static analysis techniques automatically include the information from the property, without having to take multiple formalisms into account.

A basic static analysis is already performed when encoding a model checking problem into a PBES. Intuitively, those parts of the system that do not influence validity of the property that is checked are automatically excluded from the PBES during the translation.

The main contribution in this chapter is a static analysis method for PBESs. This method consists of three separate phases. First, we compute a control flow graph for a given PBES. Second, we use this control flow graph to analyse which data parameters are relevant at which control flow locations. The final step in the method consists of assigning those data parameters that are not relevant for a control flow location some default, fixed value.

The notion of a control flow graph for PBESs is not self-evident, as a PBES does not have an obvious graph structure. Instead, the control flow is typically encoded in the parameters of the equations, which may come from both the property and the specification. An additional complication is that equations in PBESs can be mutually recursive, which means that parameters of one equation may affect parameters in another equation.

We propose a notion of a control flow parameter that allows for identifying a meaningful control flow graph of a PBES. Moreover, we provide efficient heuristics for identifying control flow parameters. Using these parameters, we define two different types of control flow graphs. The first—global—control flow graph considers all control flow parameters, and the values these can take on, simultaneously. Its size can grow exponentially in the number of control flow parameters. Drawing inspiration from [PT09], we therefore also define a second—local—type of control flow graph consisting of one graph per control flow parameter.

For both types of control flow graph we define a dedicated data flow analysis that conservatively marks data parameters that may influence the solution of the PBES. The global control flow graph permits a more fine-grained data flow analysis; the data flow analysis for the local flow graph uses a trick that permits some information to be transferred between different control flow parameters.

The markings obtained by the data flow analysis are used to reset irrelevant data parameters to a default value as soon as possible. This leads to a reduction of the size of the underlying Boolean equation system. We prove that both versions of our data flow analysis, and the consequent resetting of irrelevant data parameters, are sound, *i.e.*, they preserve the solution of the equation system, and, therefore the answer to the encoded verification problem. The soundness proof that we present relies on the notion of consistent correlations due to Willemse [Wil10], which is a generalisation of idempotence identifying bisimilarity, see Definition 3.40, to PBESs.

We implemented our reduction in the context of the mCRL2 toolset [Cra+13] and applied these to the applicable subset of examples from Chapter 5. Our results show that reductions of about 90% of the size of the underlying Boolean equation systems can be achieved.

For our analysis we draw inspiration from [GL02], which presents static analysis for state spaces in general, and [PT09] where live variable analysis is applied to reduce state spaces. In the latter, a reconstruction of the control flow is described for symbolic descriptions of processes without mutual recursion.

Liveness analysis techniques are well-known in compiler construction [ASU86] where they are used for reducing execution time. The idea of using liveness analysis techniques for state space reduction was first described by Bozga *et al.* [FBG03]. In [YG04] a similar technique was presented, using an analysis of the control flow graph.

The aforementioned techniques are restricted to an analysis of state spaces. A number of static analysis techniques, inspired by [GL02], were developed in [OWW09]. These allow the automatic reduction of the complexity of PBESs, hence also taking the property into account. In *ibid.* the authors also showed that intractable verification problems can become tractable because of their static analysis techniques. Our methods generalise these techniques.

**Structure of this chapter.** In Section 6.1 we give an introduction to the PBES theory. In Section 6.2 we describe our construction of control flow graphs for PBESs. These are used in Section 6.3 to determine live variables and reset irrelevant parameters. We present an optimisation of the analysis in Section 6.4. The approach is evaluated in Section 6.5, and we conclude in Section 6.6.

## 6.1 Parameterised Boolean Equation Systems

Parameterised Boolean equation systems are fixed point equation systems in which the equations are parameterised by abstract data types. They generalise the Boolean equation systems described in Section 2.5.

Throughout this chapter, we assume non-empty abstract data types that are represented by data sorts $D_1, D_2, \ldots$, and operations on these sorts. Let $\mathcal{D}$ be a set of sorted data variables. We write vectors in boldface, *e.g.* $\mathbf{d}$ is used to denote a vector of data variables. We write $\mathbf{d}[i]$ to denote the $i^{th}$ element of a vector $\mathbf{d}$.

A semantic set $\mathbb{D}$ is associated to every sort $D$, such that every term of sort $D$, and all operations on $D$ are mapped to the elements and operations of $\mathbb{D}$ they represent. We assume an interpretation function $[\![\_]\!]$ that maps every closed term $t$ of sort $D$ to the data element $[\![t]\!]$ that it represents. For open terms we utilise an environment $\delta$ that maps each variable from $\mathcal{D}$ to a data element of the associated type. The interpretation $[\![t]\!]\delta$ of an open term is given by $\delta(t)$, where the extension of $\delta$ to arbitrary terms is standard. Environments may be updated, such that $(\delta[v/d])(d')$ results in $v$ if $d' = d$, and $\delta(d')$ otherwise.

We specifically assume the existence of a sort $B$ with elements $\mathsf{true}$ and $\mathsf{false}$ representing the Booleans $\mathbb{B}$ and a sort $N = \{0, 1, 2, \ldots\}$ representing the natural numbers $\mathbb{N}$. For these sorts, we assume that the usual operators are available and, for readability, these are written the same as their semantic counterparts.

Before we formally define the notion of a parameterised Boolean equation system, we formalise the notion of *predicate formulae*. An example of a PBES is given in Example 6.8 on page 132.

**Definition 6.1.** *Predicate formulae* are defined through the following grammar:

$$\varphi, \psi ::= b \mid X(\mathbf{e}) \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \forall d : D.\varphi \mid \exists d : D.\varphi$$

in which $b$ is a data term of sort $B$, $X$ is a predicate variable of sort $\mathbf{D} \to B$, taken from some sufficiently large set $\mathscr{P}$ of predicate variables, and $\mathbf{e}$ is a vector of data terms of sort $\mathbf{D}$.

We assume that $\wedge$ and $\vee$ associate to the left, and that $\wedge$ binds stronger than $\vee$. Note that predicate formulae are directly in *positive form*, ensuring monotonicity.

Freely occurring data variables in $\varphi$ are denoted by $\mathsf{free}(\varphi)$. Predicate formulae without predicate variables are called *simple*. We assume that if a data variable is bound by a quantifier in a formula $\varphi$, it does not also occur free within $\varphi$.

**Definition 6.2.** The interpretation of a predicate formula $\varphi$ in the context of a predicate environment $\eta : \mathscr{P} \to \mathbb{D} \to \mathbb{B}$ and data environment $\delta$ is denoted as $[\![\varphi]\!]\eta\delta$, where:

$$[\![b]\!]\eta\delta \triangleq [\![b]\!]\delta \qquad [\![X(e)]\!]\eta\delta \triangleq \eta(X)([\![e]\!]\delta)$$

$$[\![\varphi \wedge \psi]\!]\eta\delta \triangleq [\![\varphi]\!]\eta\delta \wedge [\![\psi]\!]\eta\delta \qquad [\![\varphi \vee \psi]\!]\eta\delta \triangleq [\![\varphi]\!]\eta\delta \vee [\![\psi]\!]\eta\delta$$

$$[\![\forall d : D.\varphi]\!]\eta\delta \triangleq \forall v \in \mathbb{D}.[\![\varphi]\!]\eta\delta[v/d] \qquad [\![\exists d : D.\varphi]\!]\eta\delta \triangleq \exists v \in \mathbb{D}.[\![\varphi]\!]\eta\delta[v/d]$$

We define *logical equivalence* between two predicate formulae $\varphi, \psi$, denoted $\varphi \equiv \psi$, as $[\![\varphi]\!]\eta\delta = [\![\psi]\!]\eta\delta$ for all $\eta, \delta$.

Parameterised Boolean equation systems (PBESs), or *equation systems* for short, are sequences of fixed point equations ranging over predicate formulae.

**Definition 6.3.** Equation systems are defined by the following grammar:

$$\mathscr{E} ::= \epsilon \mid (\nu X(\mathbf{d}^X : \mathbf{D}) = \varphi)\mathscr{E} \mid (\mu X(\mathbf{d}^X : \mathbf{D}) = \varphi)\mathscr{E}$$

in which $\epsilon$ denotes the empty equation system; $\mu$ and $\nu$ are the least and greatest fixed point signs, respectively; $X$ is a sorted predicate variable of sort $\mathbf{D} \to B$, $\mathbf{d}^X$ is a vector of formal parameters, and $\varphi$ is a predicate formula.

By convention, we write $\varphi_X$ to denote the right-hand side of the defining equation for $X$ in a given equation system $\mathscr{E}$. The set of *formal parameters* of a predicate variable $X$ is denoted $\mathsf{par}(X)$ and we assume that $\mathsf{free}(\varphi_X) \subseteq \mathsf{par}(X)$. We typically omit the superscript $X$ from $\mathbf{d}^X$ if it is clear from the context.

Let $\mathsf{bnd}(\mathscr{E})$ denote the set of predicate variables occurring at the left hand sides of the equations in $\mathscr{E}$; we refer to these variables as $\mathscr{E}$'s *bound predicate variables*. In a similar way, we denote the set of *occurring predicate variables* by $\mathsf{occ}(\mathscr{E})$. The formal definitions of $\mathsf{bnd}$ and $\mathsf{occ}$ are analogous to those for proposition variables, see page 22. Throughout this chapter, we deal with equation systems that are both *well-formed* and *closed*: every bound predicate variable occurs in the left hand side of exactly one equation of $\mathscr{E}$, and all predicate variables occurring at the right-hand side are taken from the set of bound predicate variables, respectively.

To each PBES $\mathscr{E}$ we associate a *top assertion*, denoted **init** $X(\mathbf{v})$, where we require $X \in \mathsf{bnd}(\mathscr{E})$. For a parameter $\mathbf{d}[m] \in \mathsf{par}(X)$ for the top assertion **init** $X(\mathbf{v})$ we define the value $\mathsf{init}(\mathbf{d}[m])$ as $\mathbf{v}[m]$.

We next define a PBES's semantics. Let $\mathbb{B}^{\mathbb{D}}$ denote the set of functions $f : \mathbb{D} \to \mathbb{B}$, and consider the ordering $\sqsubseteq$ on its elements, defined as $f \sqsubseteq g$ iff for all $v \in \mathbb{D}$, $f(v)$ implies $g(v)$. Observe that $(\mathbb{B}^{\mathbb{D}}, \sqsubseteq)$ is a complete lattice.

An equation gives rise to a predicate transformer on this lattice as follows: a predicate formula $\varphi$ can be viewed (syntactically) as a functional $\lambda \mathbf{d}\colon \mathbf{D}.\varphi$, abbreviated by $\lambda \mathbf{d}.\varphi$. The interpretation of $\lambda \mathbf{d}.\varphi$, denoted $[\![\lambda \mathbf{d}.\varphi]\!]\eta\delta$, is the functional $(\lambda \mathbf{v} \in \mathbb{D}.[\![\varphi]\!]\eta\delta[\mathbf{v}/\mathbf{d}])$, which is a function in $\mathbb{B}^{\mathbb{D}}$. The predicate transformer associated to a functional $[\![\lambda \mathbf{d}.\varphi]\!]\eta\delta$ is given by $\lambda f \in \mathbb{B}^{\mathbb{D}}.[\![\lambda \mathbf{d}.\varphi]\!]\eta[f/X]\delta$. Since the predicate transformers defined this way are monotonic and $(\mathbb{B}^{\mathbb{D}}, \sqsubseteq)$ is a complete lattice, the extremal fixed points of these predicate transformers exist. We denote these by $\sigma f \in \mathbb{B}^{\mathbb{D}}.[\![\lambda \mathbf{d}.\varphi]\!]\eta[f/X]\delta$, for $\sigma \in \{\mu, \nu\}$. We now extend the semantics of individual equations to PBESs.

**Definition 6.4.** The *solution* of an equation system in the context of a predicate environment $\eta$ and data environment $\delta$ is defined inductively as follows:

$$[\![\epsilon]\!]\eta\delta \triangleq \eta$$
$$[\![(\sigma X(\mathbf{d}\colon \mathbf{D}) = \varphi_X)\mathscr{E}]\!]\eta\delta \triangleq [\![\mathscr{E}]\!](\eta[\sigma f \in \mathbb{B}^{\mathbb{D}}.[\![\lambda \mathbf{d}.\varphi_X]\!]([\![\mathscr{E}]\!]\eta[f/X]\delta)\delta/X])\delta$$

The solution prioritises the fixed point signs of equations that come first over the fixed point signs of equations that follow, while respecting the equations. The solution to a predicate variable in a *closed* PBES is independent of the predicate and data environments in which it is evaluated. We therefore typically leave out these environments and write $[\![\mathscr{E}]\!]$ instead of $[\![\mathscr{E}]\!]\eta\delta$ for closed equation systems.

Observe that the semantics of PBESs is similar to that of Boolean equation systems, yet it is more complicated due to the inclusion of the data types and the corresponding data environment.

For the correctness proofs of our transformation we rely on *consistent correlations*. Note that the definition presented here generalises the consistent correlations for Boolean equation systems from Definition 2.32.

The *signature* [Wil10] of a predicate variable $X$ of sort $\mathbf{D} \to B$, $\mathsf{sgt}(X)$, is the product $\{X\} \times \mathbb{D}$. The notion of signature is lifted to sets of predicate variables $P \subseteq \mathscr{P}$ in the natural way, *i.e.* $\mathsf{sgt}(P) = \bigcup_{X \in P} \mathsf{sgt}(X)$.[1]

**Definition 6.5** ([Wil10, Definition 6])**.** Let $R \subseteq \mathsf{sgt}(\mathscr{P}) \times \mathsf{sgt}(\mathscr{P})$ be an arbitrary relation. A predicate environment $\eta$ is an $R$-correlation iff $(X, \mathbf{v})\ R\ (X', \mathbf{v}')$ implies $\eta(X)(\mathbf{v}) = \eta(X')(\mathbf{v}')$.

A *block* is a non-empty equation system of like-signed fixed point equations. Given an equation system $\mathscr{E}$, a block $\mathscr{B}$ is maximal if its neighbouring equations in $\mathscr{E}$ are of a different sign than the equations in $\mathscr{B}$. The $i^{th}$ maximal block in $\mathscr{E}$ is denoted by $\mathscr{E}\rceil i$. For relations $R$ we write $\Theta_R$ for the set of $R$-correlations.

**Definition 6.6** ([Wil10, Definition 7])**.** Let $\mathscr{E}$ be an equation system. A relation $R \subseteq \mathsf{sgt}(\mathscr{P}) \times \mathsf{sgt}(\mathscr{P})$ is a *consistent correlation* on $\mathscr{E}$, if for $X, X' \in \mathsf{bnd}(\mathscr{E})$, $(X, \mathbf{v})\ R\ (X', \mathbf{v}')$ implies:

1. for all $i$, $X \in \mathsf{bnd}(\mathscr{E}\rceil i)$ iff $X' \in \mathsf{bnd}(\mathscr{E}\rceil i)$

[1] Note that in [Wil10] the notation $\mathsf{sig}$ is used to denote the signature. Here we deviate from this notation due to the naming conflict with the *significant parameters* of a formula, which also is standard notation introduced in [OWW09], and which we introduce in Section 6.3.

2. for all $\eta \in \Theta_R$, $\delta$, we have $[\![\varphi_X]\!]\eta\delta[\mathbf{v}/\mathbf{d}] = [\![\varphi'_X]\!]\eta\delta[\mathbf{v}'/\mathbf{d}']$

For $X, X' \in \mathsf{bnd}(\mathscr{E})$, we say $(X, \mathbf{v})$ and $(X', \mathbf{v}')$ consistently correlate, denoted as $(X, \mathbf{v}) \doteqdot (X', \mathbf{v}')$ iff there exists a correlation $R \subseteq \mathsf{sgt}(\mathsf{bnd}(\mathscr{E})) \times \mathsf{sgt}(\mathsf{bnd}(\mathscr{E}))$ such that $(X, \mathbf{v})\ R\ (X', \mathbf{v}')$ .

If the variables in two equation systems do not overlap we call them *compatible*. Consistent correlations can be lifted to variables in different, compatible equation systems $\mathscr{E}$ and $\mathscr{E}'$. This can, *e.g.*, be achieved by merging the equation systems to an equation system $\mathscr{F}$, in which $X \in \mathsf{bnd}(\mathscr{F}\lceil i)$ if and only if $X \in \mathsf{bnd}(\mathscr{E}\lceil i)$, and likewise for $\mathscr{E}'$. The consistent correlation can then be defined on $\mathscr{F}$.

The following theorem [Wil10] shows the relation between consistent correlations and the solution of a PBES.

**Theorem 6.7** ([Wil10, Theorem 2])**.** *Let $\mathscr{E}$, $\mathscr{E}'$ be compatible equation systems, and $\doteqdot$ a consistent correlation. Then for all $X \in \mathsf{bnd}(\mathscr{E})$, $X' \in \mathsf{bnd}(\mathscr{E}')$ and all $\eta \in \Theta_{\doteqdot}$, we have* $(X, \mathbf{v}) \doteqdot (X', \mathbf{v}') \implies [\![\mathscr{E}]\!]\eta\delta(X)(\mathbf{v}) = [\![\mathscr{E}']\!]\eta\delta(X')(\mathbf{v}')$

In Section 2.5 we have seen how a Boolean equation systems can be obtained from model checking problems. In a similar way, PBESs can be obtained from a variety of verification problems such as model checking or equivalence checking [GW05a; Che+07]. To solve a PBES, and thereby the verification problem it encodes, it is typically *instantiated* into a Boolean equation system [DPW08], using a process similar to explicit state space generation. Reducing the time spent on instantiation is therefore instrumental in speeding up solving such problems. The example below, which we use as a running example throughout this chapter, illustrates how a model checking problem can be reduced to a PBES solving problem.

**Example 6.8.** Consider the following specification of a lossy one place buffer that, when $s = 1$, can read a data element through *receive*, and then, non-deterministically (by means of the $\tau$ transitions), loses the data element when $s = 3$, or forwards the data element through *send* when $s = 4$. After this, it is back in its initial state. For messages we use a type $D$, containing at least the element $d_1$.

$$\begin{aligned}&\textbf{proc}\ P(s\colon N, d\colon D) = \textstyle\sum_{e\colon D}(s = 1) \to \mathit{receive}(e).P(2, e)\\ &\quad +(s = 2) \to \tau.P(3, d) + (s = 2) \to \tau.P(4, d)\\ &\quad +(s = 3) \to \mathit{lost}.P(1, d) + (s = 4) \to \mathit{send}(d).P(1, d_1);\\ &\textbf{init}\ P(1, d_1);\end{aligned}$$

The (first order) modal $\mu$-calculus formula below asserts that invariantly, if a message $v$ is received through *receive*, then, as long as no other message is read through *receive*, all messages delivered must match message $v$.

$$\begin{aligned}&\nu X.[\mathsf{true}]X \wedge\\ &\quad (\forall v\colon D.[\mathit{receive}(v)]\mu Y.([\overline{\exists w\colon D.\mathit{receive}(w)}]Y \wedge \forall u\colon D.[\mathit{send}(u)](v = u))).\end{aligned}$$

The model checking problem whether the lossy buffer satisfies the above formula is converted to the following PBES. Observe that in this PBES, the equation for $X$ depends on

that of $Y$: in the first conjunct of the equation for $X$, there is a recursion to the equation for $Y$ through $Y(2, e, e)$.

$$\begin{aligned}
&\nu X(s\colon N, d\colon D) = (\forall e\colon D.s = 1 \implies Y(2,e,e)) \wedge (\forall e\colon D.s = 1 \implies X(2,e))\\
&\quad \wedge (s = 2 \implies X(3,d)) \wedge (s = 2 \implies X(4,d)) \wedge (s = 3 \implies X(1,d))\\
&\quad \wedge (s = 4 \implies X(1,d_1))\\
&\nu Y(s\colon N, d, v\colon D) = (s = 4 \implies d = v) \wedge (s = 2 \implies Y(3,d,v))\\
&\quad \wedge (s = 2 \implies Y(4,d,v)) \wedge (s = 3 \implies Y(1,d,v)) \wedge (s = 4 \implies Y(1,d_1,v))\\
&\textbf{init } X(1,d_1);
\end{aligned}$$

## 6.2 Reconstructing Control Flow

Our static analysis techniques presented in the next sections are based on a notion of *control flow* in a PBES. In traditional settings, the artefacts analysed have a clear graph structure, and the notion of control flow is more or less a commonly understood concept. However, in our setting of PBESs, this is not the case. This is largely due to the fact that a PBES consists of sequences of equations over predicate formulae, for which there is no obvious graph structure.

In Section 6.2.1, we propose a notion of *control flow parameters* that permits us to define a control flow graph that is meaningful in our setting in Section 6.2.3. We describe heuristics for computing control flow graphs efficiently in Section 6.2.2.

### 6.2.1 Control Flow Parameters

The parameters of an equation in a PBES typically encode (parts of) the state space of a system and the information vital for the property that is verified in a model checking problem. Similarly, in the encoding of equivalence checking problems, the parameters of a PBES typically encode (parts of) the state spaces of both systems that are compared. It is therefore to be expected that the control of a system is reflected by the changes of values of a subset of the parameters in a PBES equation. The predicate variable instances of the form $X(\mathbf{e})$, present in the right-hand sides of the equations in a PBES, essentially dictate how the values of the parameters change.

In view of these observations, we are interested in identifying how the predicate variable instances affect the values of parameters. A complication is that there can be many different occurrences of syntactically indistinguishable predicate variable instances that, due to the context in which they are contained in a predicate formula, can be semantically different. We therefore first introduce notation to identify individual predicate variable instances in a formula.

We denote the number of predicate variable instances occurring in a predicate formula $\varphi$ by $\mathsf{npred}(\varphi)$. We assume that predicate variable instances in $\varphi$ are assigned a unique natural number between 1 and $\mathsf{npred}(\varphi)$, counting from left to right.

**Definition 6.9.** Let $\varphi$ be a predicate formula and let $i$ be between 1 and $\mathsf{npred}(\varphi)$. The functions $\mathsf{pred}(\varphi, i)$, $\mathsf{data}(\varphi, i)$ and $\mathsf{PVI}(\varphi, i)$ are such that the predicate variable instance $\mathsf{PVI}(\varphi, i)$ is the $i^{\text{th}}$ predicate variable instance in $\varphi$, syntactically present as $\mathsf{pred}(\varphi, i)(\mathsf{data}(\varphi, i))$.

We define the syntactic replacement of the predicate variable instance at position $i$ by $\psi$ in formula $\varphi$, denoted as $\varphi[i \mapsto \psi]$, as follows.

**Definition 6.10.** Let $\psi$ be a predicate formula, and let $i \leq \mathsf{npred}(\varphi)$, $\varphi[i \mapsto \psi]$ is defined inductively as follows.

$$
\begin{aligned}
b[i \mapsto \psi] &\triangleq b \\
Y(e)[i \mapsto \psi] &\triangleq \begin{cases} \psi & \text{if } i = 1 \\ Y(e) & \text{otherwise} \end{cases} \\
(\forall d : D.\varphi)[i \mapsto \psi] &\triangleq \forall d : D.\varphi[i \mapsto \psi] \\
(\exists d : D.\varphi)[i \mapsto \psi] &\triangleq \exists d : D.\varphi[i \mapsto \psi] \\
(\varphi_1 \wedge \varphi_2)[i \mapsto \psi] &\triangleq \begin{cases} \varphi_1 \wedge \varphi_2[(i - \mathsf{npred}(\varphi_1)) \mapsto \psi] & \text{if } i > \mathsf{npred}(\varphi_1) \\ \varphi_1[i \mapsto \psi] \wedge \varphi_2 & \text{if } i \leq \mathsf{npred}(\varphi_1) \end{cases} \\
(\varphi_1 \vee \varphi_2)[i \mapsto \psi] &\triangleq \begin{cases} \varphi_1 \vee \varphi_2[(i - \mathsf{npred}(\varphi_1)) \mapsto \psi] & \text{if } i > \mathsf{npred}(\varphi_1) \\ \varphi_1[i \mapsto \psi] \vee \varphi_2 & \text{if } i \leq \mathsf{npred}(\varphi_1) \end{cases}
\end{aligned}
$$

A *control flow parameter* is, intuitively, a parameter whose exact value we always know *before* and *after* recursing via a predicate variable instance. That is, we require of a control flow parameter that a recursion through the predicate variable instance is possible only when the control flow parameter has a fixed, known value, and, at the same time, we know the effect this recursion has on the value of the control flow parameter. We now make this idea more precise, by employing a collection of partial functions.

**Definition 6.11.** Let $s : \mathscr{P} \times \mathbb{N} \times \mathbb{N} \to D$, $t : \mathscr{P} \times \mathbb{N} \times \mathbb{N} \to D$, and $c : \mathscr{P} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be partial functions, where $D$ is the union of all ground data sort expressions. The triple $(s, t, c)$ is a *unicity constraint* for PBES $\mathscr{E}$ if, for all $X \in \mathsf{bnd}(\mathscr{E})$ and $1 \leq i \leq \mathsf{npred}(\varphi_X)$:

- if $s(X, i, j){=}e$ then $\varphi_X \equiv \varphi_X[i \mapsto (\mathbf{d}[j] = e \wedge \mathsf{PVI}(\varphi_X, i))]$,
- if $t(X, i, j){=}e$ then $\varphi_X \equiv \varphi_X[i \mapsto (\mathsf{data}(\varphi_X, i)[j] = e \wedge \mathsf{PVI}(\varphi_X, i))]$,
- if $c(X, i, j){=}k$ then $\varphi_X \equiv \varphi_X[i \mapsto (\mathsf{data}(\varphi_X, i)[k] = \mathbf{d}[j] \wedge \mathsf{PVI}(\varphi_X, i))]$.

The function $s$ in a unicity constraint exactly captures that, when defined for $s(X, i, j)$, the $i^{\text{th}}$ predicate variable instance in $\varphi_X$ only needs to be considered in case variable $\mathbf{d}[j]$ has the value $s(X, i, j)$. In particular, for any other value of the variable $\mathbf{d}[j]$, the truth of the predicate variable instance is immaterial to the truth of $\varphi_X$. In the same vein, when $t(X, i, j)$ is defined, then the $j^{\text{th}}$ data expression that is an argument to the $i^{\text{th}}$ predicate variable instance in $\varphi_X$ has a fixed value, given by $t(X, i, j)$. The function $c$ allows us to establish that, whenever $c(X, i, j)$ is defined to be $k$, then the value of variable $\mathbf{d}[j]$ is copied to the data expression on position $k$ in the $i^{\text{th}}$ predicate variable instance of $\varphi_X$. Note that whenever a function is not defined (denoted by $\bot$), we can draw no meaningful information from that.

**Example 6.12.** Reconsider Example 6.8. The triple $(s,t,c)$ in which $s(X,1,1) = 1$, $s(X,2,1) = 1$, $t(X,1,1) = 2$, $t(X,3,1) = 3$ and $c(X,4,2) = 2$ is a unicity constraint. By extending the mapping $c$ by defining $c(X,5,2) = 2$, $(s,t,c)$ remains a unicity constraint, but $c(X,6,2)$ may only be defined when $D$ contains only elements equal to $d_1$.

The requirements allow unicity constraints to be underspecified. In practice, it is desirable to choose the constraints as complete as possible. If, in a unicity constraint $(s,t,c)$, $s$ and $c$ are defined for a predicate variable instance, it can immediately be established that we can define $t$ as well. This is formalised by the following assumption.

**Assumption 6.13.** Let $X$ be a predicate variable, $i \leq \mathsf{npred}(\varphi_X)$, let $(s,t,c)$ be a unicity constraint, and let $e$ be a value, then

$$(s(X,i,n) = e \wedge c(X,i,n) = m) \implies t(X,i,m) = e.$$

Henceforth we assume that all unicity constraints satisfy this assumption. The overlap between $t$ and $c$ is now straightforwardly formalised in the following lemma.

**Lemma 6.14.** *Let $X$ be a predicate variable, $i \leq \mathsf{npred}(\varphi_X)$, and let $(s,t,c)$ be a unicity constraint, then if $(s(X,i,n)$ and $t(X,i,m)$ are both defined,*

$$c(X,i,n) = m \implies s(X,i,n) = t(X,i,m).$$

*Proof.* Immediately from the definitions and Assumption 6.13. □

The existence of a unicity constraint provides sufficient information for establishing a meaningful set of control flow parameters in a PBES. In the rest of this chapter we assume the existence of a unicity constraint $\mathsf{source}$, $\mathsf{dest}$ and $\mathsf{copy}$. We characterise the set of control flow parameters by imposing additional restrictions on the unicity constraints.

We incrementally construct the set of control flow parameters by first locally establishing requirements on parameters. This set is then further restricted by requirements that take the global structure of the equation system into account.

Locally, a parameter is a control flow parameter if, in every self-recursion, it is either copied, or a destination value is known.

**Definition 6.15.** Let $\mathscr{E}$ be a PBES, with $X \in \mathsf{bnd}(\mathscr{E})$. Parameter $\mathbf{d}[n] \in \mathsf{par}(X)$ is a *local control flow parameter (LCFP)* if for all $i$ such that $\mathsf{pred}(\varphi_X, i) = X$, either $\mathsf{source}(X,i,n)$ and $\mathsf{dest}(X,i,n)$ are defined, or $\mathsf{copy}(X,i,n) = n$.

Towards defining a set of control flow parameters, we add a global consistency requirement on LCFPs. A set of LCFPs is globally consistent if every control flow parameter is only ever set to a constant, or the value of another control flow parameter.

**Definition 6.16.** Let $\mathscr{E}$ be a PBES. A set $\mathscr{C}$ of LCFPs is *globally consistent* if, for all $X \in \mathsf{bnd}(\mathscr{E})$, and all $\mathbf{d}^X[n] \in \mathsf{par}(X)$, we have for all $Y \in \mathsf{bnd}(\mathscr{E}) \setminus \{X\}$ and all $i$ such that $\mathsf{pred}(\varphi_Y, i) = X$, either $\mathsf{dest}(Y,i,n)$ is defined, or $\mathsf{copy}(Y,i,m) = n$ for some LCFP $\mathbf{d}^Y[m] \in \mathsf{par}(Y)$.

*Remark* 6.17. Control flow parameters and data parameters can be separated completely by adding the following requirement to Definition 6.16: for all $\mathbf{d}^X[n] \in \mathsf{par}(X)$, we have for all $i$ such that $\mathsf{pred}(\varphi_X, i) = Y \neq X$, whenever $\mathbf{d}^X[n]$ $\mathsf{affects}$ $\mathsf{data}(\varphi_X, i)[m]$ then $\mathbf{d}^Y[m] \in \mathscr{C}$.

Additionally, a control flow parameter for an equation is not allowed to (indirectly) influence other control flow parameters in the same equation. We formalise this in two steps by relating control flow parameters.

**Definition 6.18.** Let $\mathscr{E}$ be a PBES. With $\mathscr{C}$ a set of globally consistent LCFPs. We say that $\mathbf{d}^X[n], \mathbf{d}^Y[m] \in \mathscr{C}$ are *related*, denoted $\mathbf{d}^X[n] \sim \mathbf{d}^Y[m]$, if either $\mathsf{pred}(X,i) = Y$, $m = \mathsf{copy}(X,i,n)$ and $\mathsf{dest}(X,i,m)$ is undefined, or $\mathsf{pred}(Y,i') = X$, $n = \mathsf{copy}(Y,i',m)$ and $\mathsf{dest}(Y,i',n)$ is undefined for some $i, i'$.

The relation $\sim$ relates control flow parameters that syntactically influence each other, and are therefore related. For developing effective heuristics we often need to be able to relate control flow parameters that only 'influence' each other by passing constants. Any relation that we can construct has to satisfy the following consistency requirement.

**Definition 6.19.** Let $\mathscr{C}$ be a set of globally consistent LCFPs and let $\approx$ be an arbitrary equivalence relation on $\mathscr{C}$ satisfying $\sim^* \subseteq \approx$. Then the pair $\langle \mathscr{C}, \approx \rangle$ defines a set of *control flow parameters (CFPs)* if for all $X \in \mathsf{bnd}(\mathscr{E})$ and all $d, d' \in \mathscr{C} \cap \mathsf{par}(X)$, if $d \approx d'$, then $d = d'$.

The set $[d]_\approx$, defined as $\{d' \in \mathscr{C} \mid d \approx d'\}$ is the set of *identical* control flow parameters.

Useful heuristics in constructing the relation $\approx$ in the definition above are relating parameters with the same names in different equations, or relating parameters with the same positions in different equations.

Given a set of CFPs $\langle \mathscr{C}, \approx \rangle$ and a control flow parameter $c$, $\mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(c)$ produces a variable that represents $c$'s equivalence class, such that for CFPs $c, c'$, if $c \approx c'$, then $\mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(c) = \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(c')$. We generalise $\mathsf{repr}$ to sequences of CFPs in the obvious way.

We say that a set of CFPs $\langle \mathscr{C}, \approx \rangle$ is *meaningful* for $\mathscr{E}$ if, for every $c \in \mathscr{C}$, there exists some $\mathbf{c}^X[n]$ s.t. $c \approx \mathbf{c}^X[n]$, for which either a value is assigned in the initial value of the PBES, or for which there exist $X, i$ such that $\mathsf{dest}(X,i,n)$ is defined. This ensures that for a set of meaningful CFPs, we can find at least one value in $\mathsf{values}(\mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(c))$ after the following transformation.

Definition 6.19 ensures that we can transform our PBES in such a way that every equation has the same control flow parameters. We formalise this transformation.

**Definition 6.20.** Let $\mathscr{E} \triangleq (\sigma_1 X_1(\mathbf{c}^{X_1}, \mathbf{d}^{X_1}) = \varphi_{X_1}) \cdots (\sigma_n X_n(\mathbf{c}^{X_n}, \mathbf{d}^{X_n}) = \varphi_{X_n})$, with a meaningful set of CFPs $\langle \mathscr{C}, \approx \rangle$ where $\mathbf{c}^{X_i}$ are sequences of CFPs in $\mathscr{C}$, and $\mathbf{d}^{X_i}$ are sequences of data parameters. We unify the control flow parameters in $\mathscr{E}$ using $\mathsf{unify}$, which is defined inductively as follows.

$$\mathsf{unify}(\epsilon) = \epsilon$$
$$\mathsf{unify}((\sigma X(\mathbf{c}^X, \mathbf{d}^X) = \varphi)\mathscr{E}) = (\sigma \bar{X}(\mathbf{c}', \mathbf{d}^X) = \mathsf{unify}_X(\varphi))\mathsf{unify}(\mathscr{E})$$

Here $\mathbf{c}'$ is the sequence of variables containing exactly the representative for each equivalence class in $\langle \mathscr{C}, \approx \rangle$. For formulae, $\mathsf{unify}$ is defined inductively as follows.

$$\mathsf{unify}_X(b) = b[\mathbf{c}^X := \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^X)] \qquad \mathsf{unify}_X(Y(\mathbf{v}, \mathbf{w})) = \bar{Y}(\mathbf{v}', \mathbf{w}')$$
$$\mathsf{unify}_X(\varphi \wedge \psi) = \mathsf{unify}_X(\varphi) \wedge \mathsf{unify}_X(\psi) \qquad \mathsf{unify}_X(\varphi \vee \psi) = \mathsf{unify}_X(\varphi) \vee \mathsf{unify}_X(\psi)$$
$$\mathsf{unify}_X(\forall d\colon D.\varphi) = \forall d\colon D.\mathsf{unify}_X(\varphi) \qquad \mathsf{unify}_X(\exists d\colon D.\varphi) = \exists d\colon D.\mathsf{unify}_X(\varphi)$$

Here $\mathbf{w}' = \mathbf{w}[\mathbf{c}^X := \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^X)]$, and $\mathbf{v}'$ in the unification of predicate variable instance ensures that every new control flow parameter gets the value of the CFP that it represents, if any, and otherwise it is copied. Positionally $\mathbf{v}'$ is defined as follows.

$$\mathbf{v}'[i] = \begin{cases} \mathbf{v}[j][\mathbf{c}^X := \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^X)] & \text{if } \mathbf{c}'[i] = \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^Y[j]) \\ \mathbf{c}'[i] & \text{otherwise} \end{cases}$$

In the unification of the initial value of a PBES, in the definition of $\mathbf{v}'[i]$, the value $\min\{\mathsf{values}(\mathbf{c}'[i])\}$ is introduced instead of $\mathbf{c}'[i]$.

This transformation preserves the solution of the equation system, as indicated by the following theorem.

**Theorem 6.21.** *Let $\mathscr{E}$ be a closed PBES, with set of CFPs $\langle \mathscr{C}, \approx \rangle$, then for all $X$, $\mathbf{v}$, $\mathbf{v}'$ and $\mathbf{w}$: $[\![\mathscr{E}]\!](X([\![\mathbf{v}]\!], [\![\mathbf{w}]\!]) = [\![\mathsf{unify}(\mathscr{E})]\!](\bar{X}([\![\mathbf{v}']\!], [\![\mathbf{w}]\!])$, provided that $\mathbf{v}'$ satisfies that $[\![\mathbf{v}'[i]]\!] = [\![\mathbf{v}[j]]\!]$ if $\mathbf{c}'[i] = \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^X[j])$.*

*Proof.* We prove this theorem by giving a relation $R$, and showing that it is a consistent correlation. The statement then follows immediately.

Let $R$ be the relation such that $(X, [\![\mathbf{v}]\!], [\![\mathbf{w}]\!])$ $R$ $(\bar{X}, [\![\mathbf{v}']\!], [\![\mathbf{w}]\!])$ for all $X$, $\mathbf{v}$, $\mathbf{v}'$, $\mathbf{w}$, provided that $\mathbf{v}'$ satisfies that $[\![\mathbf{v}'[i]]\!] = [\![\mathbf{v}[j]]\!]$ if $\mathbf{c}'[i] = \mathsf{repr}_{\langle \mathscr{C}, \approx \rangle}(\mathbf{c}^X[j])$. Relation $R$ is a consistent correlation. □

Furthermore, control flow parameters are indeed preserved by the transformation.

**Proposition 6.22.** *Let $\mathscr{E}$ be an equation system with a meaningful set of CFPs $\langle \mathscr{C}, \approx \rangle$, and let $\mathsf{unify}(\mathscr{E})$ be such that $\mathbf{c}'$ are the parameters that represent the control flow parameters in $\mathscr{C}$. For every $i$, $\mathbf{c}'[i]$ is a meaningful CFP in $\mathsf{unify}(\mathscr{E})$.*

Proof of this proposition follows by induction on the structure of the PBES. Again using a simple inductive argument, we can prove that for a set of meaningful control flow parameters, all values that the control flow parameters can attain lie within the set $\mathsf{values}$.

**Proposition 6.23.** *Let $\mathscr{E}$ be a PBES, with a meaningful set of CFPs, and let $\mathscr{F} = \mathsf{unify}(\mathscr{E}) = (\sigma_1 \bar{X}_1(\mathbf{c}', \mathbf{d}^{\bar{X}_1}) = \varphi_{\bar{X}_1}) \cdots (\sigma_n \bar{X}_n(\mathbf{c}', \mathbf{d}^{\bar{X}_n}) = \varphi_{\bar{X}_n})$. If for all $i$, $\mathbf{c}'[i]$ there is at least one $\bar{X}, j$ such that $\mathsf{dest}(\varphi_{\bar{X}}, j, i)$ is defined, or the initial value of $\mathbf{c}'[i]$ is a constant, then all reachable values of $\mathbf{c}'$ are in $\mathsf{values}(\mathbf{c}')$.*

Given the above considerations, and to simplify the presentation in the rest of this chapter, we make the following assumption.

**Assumption 6.24.** The set of control flow parameters is the same for every equation in a PBES; that is, for all $X, Y \in \mathsf{bnd}(\mathscr{E})$ in a PBES $\mathscr{E}$ we have $d \in \mathsf{par}(X)$ is a CFP iff $d \in \mathsf{par}(Y)$ is a CFP.

**Example 6.25.** The unicity constraint of Example 6.12 can be extended in such a way that the parameters $(X, s)$ and $(Y, s)$ satisfy all requirements of CFP, and hence are both control flow parameters. Note that there is no unicity constraint that enables us to mark parameters $(X, d)$, $(Y, v)$ and $(Y, d)$ as control flow parameters, as these always violate the second requirement of a CFP. $(X, s)$ and $(Y, s)$ may be related by some $\approx$, as this will not violate Definition 6.19.

Henceforth, any parameter that is not a control flow parameter is called a *data parameter*. For ease of reasoning, we make this distinction explicit by partitioning $\mathcal{D}$ into $\mathcal{D}^{\mathrm{CFP}}$ and $\mathcal{D}^{\mathrm{DP}}$, containing the control flow parameters and the data parameters respectively. We occasionally write equations as $\sigma X(\mathbf{c}\colon \mathbf{C}, \mathbf{d}^X\colon \mathbf{D}^X) = \varphi_X(\mathbf{c})(\mathbf{d}^X)$, where $\mathbf{c}$ are the CFPs, and $\mathbf{d}^X$ are the DPs of the equation for $X$. Observe that $\mathbf{c}$ is not superscripted (in line with Assumption 6.24). If the equation $X$ is clear from the context, we also omit the superscript of $\mathbf{d}$.

### 6.2.2 Heuristics

The control flow parameters are not necessarily efficiently or effectively computable. We therefore look for cheap heuristics that permit us to build the unicity constraints. For this, we analyse the syntactic conditions present in the predicate formulae that allow us to approximate under which conditions predicate variable instances are still relevant to the truth of the predicate formula. Such conditions, which we refer to as *guards*, are subsequently used to heuristically determine a good unicity constraint. These guards are defined as follows.

**Definition 6.26.** Let $\varphi$ be a predicate formula. We define the *guard* of predicate variable instance $\mathsf{PVI}(\varphi, i)$ for $i \leq \mathsf{npred}(\varphi)$ inductively as follows:

$$
\begin{aligned}
\mathsf{guard}^i(b) &= \mathsf{false}\\
\mathsf{guard}^i(Y) &= \mathsf{true}\\
\mathsf{guard}^i(\forall d\colon D.\varphi) &= \mathsf{guard}^i(\varphi)\\
\mathsf{guard}^i(\exists d\colon D.\varphi) &= \mathsf{guard}^i(\varphi)\\
\mathsf{guard}^i(\varphi \wedge \psi) &= \begin{cases} s(\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \text{if } i > \mathsf{npred}(\varphi)\\ s(\psi) \wedge \mathsf{guard}^i(\varphi) & \text{if } i \leq \mathsf{npred}(\varphi) \end{cases}\\
\mathsf{guard}^i(\varphi \vee \psi) &= \begin{cases} ns(\varphi) \wedge \mathsf{guard}^{i-\mathsf{npred}(\varphi)}(\psi) & \text{if } i > \mathsf{npred}(\varphi)\\ ns(\psi) \wedge \mathsf{guard}^i(\varphi) & \text{if } i \leq \mathsf{npred}(\varphi) \end{cases}
\end{aligned}
$$

where

$$
s(\varphi) = \begin{cases} \varphi & \text{if } \mathsf{npred}(\varphi) = 0\\ \mathsf{true} & \text{otherwise} \end{cases} \qquad ns(\varphi) = \begin{cases} \neg\varphi & \text{if } \mathsf{npred}(\varphi) = 0\\ \mathsf{true} & \text{otherwise} \end{cases}
$$

Intuitively, the value of a predicate variable instance $\mathsf{PVI}(\varphi, i)$ in a formula $\varphi$ is irrelevant if $\mathsf{guard}^i(\varphi)$ is unsatisfiable. This is formalised in the lemmata below.

To show that, indeed, we compute a guard, we first show that we can guard every predicate variable instance with its guard, without changing the solution.

**Lemma 6.27.** *Let $\varphi$ be a predicate formula, and let $i \leq \mathsf{npred}(\varphi)$, then for every predicate environment $\eta$ and data environment $\delta$,*

$$
[\![\varphi]\!]\eta\delta = [\![\varphi[i \mapsto (\mathsf{guard}^i(\varphi) \implies \mathsf{PVI}(\varphi, i))]]\!]\eta\delta.
$$

*Proof.* Let $\eta$ and $\delta$ be arbitrary. We proceed by induction on $\varphi$. The base cases where $\varphi = b$ and $\varphi = Y(\mathbf{e})$ are trivial, and $\forall d : D.\psi$ and $\exists d : D.\psi$ follow immediately from the induction hypothesis. We describe the case where $\varphi = \varphi_1 \wedge \varphi_2$ in detail, the $\varphi = \varphi_1 \vee \varphi_2$ is completely analogous.

Assume that $\varphi = \varphi_1 \wedge \varphi_2$. Let $i \leq \mathsf{npred}(\varphi_1 \wedge \varphi_2)$. Without loss of generality assume that $i \leq \mathsf{npred}(\varphi_1)$, the other case is analogous. According to the induction hypothesis,

$$[\![\varphi_1]\!]\eta\delta = [\![\varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \implies \mathsf{PVI}(\varphi_1, i))]]\!]\eta\delta \tag{6.1}$$

We distinguish two cases.

- $\mathsf{occ}(\varphi_2) \neq \emptyset$. Then $[\![\mathsf{guard}^i(\varphi_1)]\!]\delta\eta = [\![\mathsf{guard}^i(\varphi_1 \wedge \varphi_2)]\!]\delta\eta$ according to the definition of guard. Since $i \leq \mathsf{npred}(\varphi_1)$, we find that $[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta = [\![(\varphi_1 \wedge \varphi_2)[i \mapsto (\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) \implies \mathsf{PVI}(\varphi_1 \wedge \varphi_2, i))]]\!]\eta\delta$.
- $\mathsf{occ}(\varphi_2) = \emptyset$. We have to show that

  $$[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta = [\![\varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) \implies \mathsf{PVI}(\varphi_1, i))] \wedge \varphi_2]\!]\eta\delta$$

  From the semantics, it follows that $[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta = [\![\varphi_1]\!]\eta\delta \wedge [\![\varphi_2]\!]\eta\delta$. Combined with (6.1), and an application of the semantics, this yields

  $$[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta = [\![\varphi_1[i \mapsto (\mathsf{guard}^i(\varphi_1) \implies \mathsf{PVI}(\varphi_1, i))] \wedge \varphi_2]\!]\eta\delta.$$

  According to the definition of guard, $\mathsf{guard}^i(\varphi_1 \wedge \varphi_2) = \varphi_2 \wedge \mathsf{guard}^i(\varphi_1)$. Since $\varphi_2$ is present in the context, using monotonicity and an application of the semantics, the desired result follows. □

We can generalise the above, and guard every predicate variable instance in a formula with its guard, which preserves the solution of the formula. To this end we introduce the function guarded.

**Definition 6.28.** Let $\varphi$ be a predicate formula, then

$$\mathsf{guarded}(\varphi) \triangleq \varphi[i \mapsto (\mathsf{guard}^i(\varphi) \implies \mathsf{PVI}(\varphi, i))]_{i \leq \mathsf{npred}(\varphi)}$$

where $[i \mapsto \psi_i]_{i \leq \mathsf{npred}(\varphi)}$ is the simultaneous syntactic substitution of all $\mathsf{PVI}(\varphi, i)$ with $\psi_i$.

The following corollary follows immediately from Lemma 6.27.

**Corollary 6.29.** *For all formulae $\varphi$, and for all predicate environments $\eta$, and data environments $\delta$, $[\![\varphi]\!]\eta\delta = [\![\mathsf{guarded}(\varphi)]\!]\eta\delta$*

This corollary confirms our intuition that indeed the guards we compute effectively guard the recursions in a formula.

A good heuristic for defining the unicity constraints is by looking for positive occurrences of constraints of the form $d = e$ in the guards, where $d$ is a parameter and $e$ a constant; these can be used to define the source function. For determining the dest function, one can replace a data parameter by the value dictated by source for this parameter for a predicate variable instance and check whether data expressions in a recursion reduce to a constant under this substitution. The copy function can be defined through simple syntactic checks that determine which data expressions in a predicate variable instance consist of data parameters only. In case a variable reappears in multiple data expressions in the predicate variable instance, the copy function is left undefined.

### 6.2.3 Control Flow Graph

We next construct a control flow graph that describes how the values of the control flow parameters are affected by the predicate variable instances that occur in the predicate formulae of the PBES. The edge relation is determined using the unicity constraint $(\mathsf{source}, \mathsf{dest}, \mathsf{copy})$ that witnesses the set of control flow parameters of the PBES. The locations are determined by the possible values that the control flow parameters can assume.

**Definition 6.30.** Let $\mathbf{c}[k]$ be a control flow parameter. The set of values $\mathbf{c}[k]$ can attain, denoted $\mathsf{values}(\mathbf{c}[k])$, is defined as:

$$\{\mathsf{init}(\mathbf{c}[k])\} \cup \{v \mid \exists i \in \mathbb{N} : \exists X \in \mathsf{bnd}(\mathscr{E}) : \mathsf{source}(X,i,k) = v \vee \mathsf{dest}(X,i,k) = v\}.$$

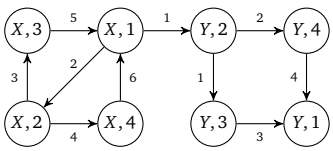
Note that $|\mathsf{values}(\mathbf{c}[k])|$ is finite. We generalise $\mathsf{values}$ to vectors of control flow parameters in the obvious way. Using the set of attainable values for the control flow parameters, we construct a control flow graph.

**Definition 6.31.** Let $\mathscr{E}$ be a PBES with control flow parameters $\mathbf{c}$, witnessed by unicity constraint $(\mathsf{source}, \mathsf{dest}, \mathsf{copy})$. The control flow graph (CFG) of $\mathscr{E}$ is a graph $(V, \longrightarrow)$ with:

- $V = \mathsf{bnd}(\mathscr{E}) \times \mathsf{values}(\mathbf{c})$, and
- $\longrightarrow \subseteq V \times \mathbb{N} \times V$ is the least relation for which, if $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{w})$ then for every $k$ either:
  - $\mathsf{source}(X,i,k) = \mathbf{v}[k]$ and $\mathsf{dest}(X,i,k) = \mathbf{w}[k]$, or
  - $\mathsf{source}(X,i,k) = \bot$, $\mathsf{copy}(X,i,k) = k$ and $\mathbf{v}[k] = \mathbf{w}[k]$, or
  - $\mathsf{source}(X,i,k) = \bot$, and $\mathsf{dest}(X,i,k) = \mathbf{w}[k]$.

Henceforth, we refer to the vertices in the control flow graph as control flow locations, or *locations*, for short.

**Example 6.32.** Reconsider the PBES from Example 6.8 and its control flow parameters, determined in Example 6.25. Its control flow graph is depicted below.




Control flow graphs are complete in the sense that all predicate variable instances that can influence the solution of $\varphi_X$ at location $\mathbf{v}$ are neighbours of the vertex $(X, \mathbf{v})$ in control flow graphs.

**Lemma 6.33.** *Let $\mathscr{E}$ be a PBES with control flow graph $(V, \longrightarrow)$. Then for all $(X, \mathbf{v}) \in V$ and all predicate environments $\eta, \eta'$ and data environments $\delta$:*

$$[\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}] = [\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta'\delta[[\![\mathbf{v}]\!]/\mathbf{c}]$$

*provided that $\eta(Y)(\mathbf{w}) = \eta'(Y)(\mathbf{w})$ for all $(Y, \mathbf{w})$ satisfying $(X, \mathbf{v}) \xrightarrow{i} (Y, \mathbf{w})$.*

*Proof.* Let $\eta, \eta'$ be predicate environments and $\delta$ be a data environment, and let $(X, \mathbf{v}) \in V$. Suppose that for all $(Y, \mathbf{w})$ for which $(X, \mathbf{v}) \xrightarrow{i} (Y, \mathbf{w})$, we know that $\eta(Y)(\mathbf{w}) = \eta'(Y)(\mathbf{w})$.

Towards a contradiction, let $[\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}] \neq [\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta'\delta[[\![\mathbf{v}]\!]/\mathbf{c}]$, then there must be a predicate variable instance $\mathsf{PVI}(\varphi_X, i,)$ such that

$$\begin{array}{rl} & \eta(\mathsf{pred}(\varphi_X, i))([\![\mathsf{data}(\varphi_X, i)]\!]\delta[[\![\mathbf{v}]\!]/\mathbf{c}]) \\ \neq & \eta'(\mathsf{pred}(\varphi_X, i))([\![\mathsf{data}(\varphi_X, i)]\!]\delta[[\![\mathbf{v}]\!]/\mathbf{c}]). \end{array} \tag{6.2}$$

Let $\mathsf{data}(\varphi_X, i) = (\mathbf{e}, \mathbf{e}')$, where $\mathbf{e}$ are the values of the control flow parameters, and $\mathbf{e}'$ are the values of the data parameters.

Consider an arbitrary control flow parameter $\mathbf{c}[\ell]$. We distinguish two cases:

- $\mathsf{source}(X, i, \ell) \neq \bot$. Then we know $\mathsf{dest}(X, i, \ell) \neq \bot$, and the requirement for the edge $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{e})$ is satisfied for $\ell$.

- $\mathsf{source}(X, i, \ell) = \bot$. Since $\mathbf{c}[\ell]$ is a control flow parameter, we can distinguish two cases based on Definitions 6.15 and 6.16:

  - $\mathsf{dest}(X, i, \ell) \neq \bot$. Then parameter $\ell$ immediately satisfies the requirements that show the existence of the edge $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{e})$ in the third clause in the definition of CFG.
  - $\mathsf{copy}(X, i, \ell) = \ell$. According to the definition of $\mathsf{copy}$, we now know that $\mathbf{v}[\ell] = \mathbf{e}[\ell]$, hence the edge $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{e})$ exists according to the second requirement in the definition of CFG.

Since we have considered an arbitrary $\ell$, we know that for all $\ell$ the requirements are satisfied, hence $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{e})$. Then according to the definition of $\eta$ and $\eta'$, $\eta(\mathsf{pred}(\varphi_X, i))([\![\mathbf{e}]\!]\delta[[\![\mathbf{v}]\!]/\mathbf{c}]) = \eta'(\mathsf{pred}(\varphi_X, i))([\![\mathbf{e}]\!]\delta[[\![\mathbf{v}]\!]/\mathbf{c}])$. This contradicts assumption (6.2), hence we find that $[\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}] = [\![\varphi_X(\mathbf{c})(\mathbf{d})]\!]\eta'\delta[[\![\mathbf{v}]\!]/\mathbf{c}]$. □

## 6.3 Data Flow Analysis

We now formalised the notion of a control flow parameter, and established heuristics to determine those parameters. Next we analyse the flow of data within an equation system. Intuitively, a data parameter $d$ of $X$ is potentially relevant if it can influence the truth of $\varphi_X$. The influence is direct if $d$ occurs in a Boolean expression in $\varphi_X$. Such parameters are called *significant* [OWW09]; they can be determined as follows.

$$\begin{array}{rlrl} \mathsf{sig}(b) & \triangleq \mathsf{free}(b) & \mathsf{sig}(Y(e)) & \triangleq \emptyset \\ \mathsf{sig}(\varphi \wedge \psi) & \triangleq \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi) & \mathsf{sig}(\varphi \vee \psi) & \triangleq \mathsf{sig}(\varphi) \cup \mathsf{sig}(\psi) \\ \mathsf{sig}(\exists d : D.\varphi) & \triangleq \mathsf{sig}(\varphi) \setminus \{d\} & \mathsf{sig}(\forall d : D.\varphi) & \triangleq \mathsf{sig}(\varphi) \setminus \{d\} \end{array}$$

Indirect influences are the result of parameters affecting the value of other parameters through predicate variable instances.

**Definition 6.34.** Let $X(\mathbf{e})$ be a predicate variable instance. We say that a variable $d$ affects $\mathbf{e}[i]$ if $d \in \mathsf{free}(\mathbf{e}[i])$.

Suppose we have a function simplify that converts $\varphi$ into an equivalent formula with lower or equal number of significant parameters; *i.e.*, $\mathsf{simplify}(\varphi) \equiv \varphi$, and $\mathsf{sig}(\varphi) \supseteq \mathsf{sig}(\mathsf{simplify}(\varphi))$. Typically simplify can be implemented by rewrite rules.

First, observe that if we assign values to some parameters, the likelihood that simplify can reduce the number of significant parameters in a formula increases. Second, observe that we have a good set of candidate parameters for which we know they can assume only a finite set of values: the control flow parameters of the previous section. In other words, given a location $(X, \mathbf{v})$ in a control flow graph, we can better approximate the set of significant parameters by considering $\mathsf{simplify}(\varphi_X[\mathbf{c} := \mathbf{v}])$. This will be the basis for our analysis. Then, through a backwards reachability using our control flow graph, we identify parameters that indirectly influence the values of significant parameters in the locations of the graph.

The above informal exposition is formalised in the definition below; the set $M$ that is constructed approximates the set of parameters that are potentially relevant in a location. Parameters that end up in $M$ for some location are not guaranteed to be relevant, and parameters that do not end up in $M$ for some location are guaranteed to be irrelevant.

**Definition 6.35.** Let $\mathscr{E}$ be a PBES and let $(V, \longrightarrow)$ be its control flow graph. We define marking $M : V \to \mathbb{P}(\mathscr{D}^{\mathrm{DP}})$ inductively as follows:

$$
\begin{aligned}
M^0(X, \mathbf{v}) \quad &= \mathsf{sig}(\mathsf{simplify}(\varphi_X[\mathbf{c} := \mathbf{v}])) \\
M^{n+1}(X, \mathbf{v}) \quad &= M^n(X, \mathbf{v}) \\
&\quad \cup \{d \in \mathsf{par}(X) \mid \exists i \in \mathbb{N}, (Y, \mathbf{w}) \in V : (X, \mathbf{v}) \xrightarrow{i} (Y, \mathbf{w}) \\
&\qquad \wedge \exists \mathbf{d}[\ell] \in M^n(Y, \mathbf{w}) : \ d \text{ affects } \mathsf{data}(\varphi_X, i)[\ell]\} \\
M(X, \mathbf{v}) \quad &= \textstyle\bigcup_{n \in \mathbb{N}} M^n(X, \mathbf{v})
\end{aligned}
$$

**Example 6.36.** Analysing our running example using the control flow graph of Example 6.32 we find that, initially, data parameters $d$ and $v$ are marked in vertex $(Y, 4)$ only. In the next step the same parameters are marked in vertex $(Y, 2)$ due to the predicate variable instance at index 2. This is also the final marking.

The syntactic marking from Definition 6.35 induces a relation $R^M$ on signatures as follows.

**Definition 6.37.** Let $M : V \to \mathbb{P}(\mathscr{D}^{\mathrm{DP}})$ be a marking. Every marking $M$ induces a relation $R^M$ such that $(X, [\![\mathbf{v}]\!], [\![\mathbf{w}]\!]) R^M (Y, [\![\mathbf{v}']\!], [\![\mathbf{w}']\!])$ if and only if $X = Y$, $[\![\mathbf{v}]\!] = [\![\mathbf{v}']\!]$, and $\forall \mathbf{d}[k] \in M(X, \mathbf{v}) : [\![\mathbf{w}[k]]\!] = [\![\mathbf{w}'[k]]\!]$.

Observe that the relation $R^M$ allows for relating *all* instances of the non-marked data parameters at a given control flow location. We prove that, if locations are related using

the relation $R^M$, then the corresponding instances in the PBES have the same solution by showing that $R^M$ is a consistent correlation.

In order to prove this, we first show that given a predicate environment and two data environments, if the solution of a formula differs between those environments, and all predicate variable instances in the formula have the same solution, then there must be a significant parameter $d$ in the formula that gets a different value in the two data environments.

**Lemma 6.38.** *For all formulae $\varphi$, predicate environments $\eta$, and data environments $\delta, \delta'$, if we know that $[\![\varphi]\!]\eta\delta \neq [\![\varphi]\!]\eta\delta'$ and $[\![\mathsf{PVI}(\varphi, i)]\!]\eta\delta = [\![\mathsf{PVI}(\varphi, i)]\!]\eta\delta'$ for all $i \leq \mathsf{npred}(\varphi)$, then $\exists d \in \mathsf{sig}(\varphi) : \delta(d) \neq \delta'(d)$.*

*Proof.* We proceed by induction on $\varphi$.

- $\varphi = b$. Trivial.
- $\varphi = Y(e)$. In this case the two preconditions contradict, and the result trivially follows.
- $\varphi = \forall e\colon D.\psi$. Assume that $[\![\forall e\colon D.\psi]\!]\eta\delta \neq [\![\forall e\colon D.\psi]\!]\eta\delta'$, and furthermore, $\forall i \leq \mathsf{npred}(\forall e\colon D.\psi) : [\![\mathsf{PVI}(\forall e\colon D.\psi, i)]\!]\eta\delta = [\![\mathsf{PVI}(\forall e\colon D.\psi, i)]\!]\eta\delta'$.

  According to the semantics, $\forall u \in \mathbb{D}.[\![\psi]\!]\eta\delta[u/e] \neq \forall u' \in \mathbb{D}.[\![\psi]\!]\eta\delta'[u'/e]$, so $\exists u \in \mathbb{D}$ such that $[\![\psi]\!]\eta\delta_1[u/e] \neq [\![\psi]\!]\eta\delta_2[u/e]$. Choose an arbitrary such $u$. Observe that also for all $i \leq \mathsf{npred}(\psi)$, we know that $[\![\mathsf{PVI}(\psi, i)]\!]\eta\delta[u/e] = [\![\mathsf{PVI}(\psi, i)]\!]\eta\delta'[u/e]$. According to the induction hypothesis, $\exists d \in \mathsf{sig}(\psi)$ such that $\delta[u/e](d) \neq \delta'[u/e](d)$. Choose such a $d$, and observe that $d \neq e$ since otherwise $u \neq u$, hence $d \in \mathsf{sig}(\forall e\colon D.\psi)$, which is the desired result.
- $\varphi = \exists e\colon D.\psi$. Analogous to the previous case.
- $\varphi = \varphi_1 \wedge \varphi_2$. Assume that $[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta \neq [\![\varphi_1 \wedge \varphi_2]\!]\eta\delta'$, and suppose that that for all $i \leq \mathsf{npred}(\varphi_1 \wedge \varphi_2)$, we know that $[\![\mathsf{PVI}(\varphi_1 \wedge \varphi_2, i)]\!]\eta\delta = [\![\mathsf{PVI}(\varphi_1 \wedge \varphi_2, i)]\!]\eta\delta'$. According to the first assumption, either $[\![\varphi_1]\!]\eta\delta \neq [\![\varphi_1]\!]\eta\delta'$, or $[\![\varphi_2]\!]\eta\delta \neq [\![\varphi_2]\!]\eta\delta'$.

  Without loss of generality, assume that $[\![\varphi_1]\!]\eta\delta \neq [\![\varphi_1]\!]\eta\delta'$, the other case is completely analogous. Observe that from our second assumption it follows that $\forall i \leq \mathsf{npred}(\varphi_1) : [\![\mathsf{PVI}(\varphi_1, i)]\!]\eta\delta = [\![\mathsf{PVI}(\varphi_1, i)]\!]\eta\delta'$. According to the induction hypothesis, we now find some $d \in \mathsf{sig}(\varphi_1)$ such that $\delta(d) \neq \delta'(d)$. Since $\mathsf{sig}(\varphi_1) \subseteq \mathsf{sig}(\varphi_1 \wedge \varphi_2)$, our result follows.
- $\varphi = \varphi_1 \vee \varphi_2$. Analogous to the previous case. □

This is now used in proving the following proposition, that shows that related signatures have the same solution. This result follows from the fact that $R^M$ is a consistent correlation.

**Proposition 6.39.** *Let $\mathscr{E}$ be a PBES, with global control flow graph $(V, \rightarrow)$, and marking $M$. For all predicate environments $\eta$ and data environments $\delta$,*

$$(X, [\![\mathbf{v}]\!], [\![\mathbf{w}]\!])R^M(Y, [\![\mathbf{v}']\!], [\![\mathbf{w}']\!]) \implies [\![\mathscr{E}]\!]\eta\delta(X(\mathbf{v}, \mathbf{w})) = [\![\mathscr{E}]\!]\eta\delta(Y(\mathbf{v}', \mathbf{w}')).$$

*Proof.* We show that $R^M$ is a consistent correlation. The result then follows immediately from Theorem 6.7.

Let $n$ be the smallest number such that for all $X, \mathbf{v}$, $M^n(X,\mathbf{v}) = M^n(X,\mathbf{v})$, and hence $M^n(X,\mathbf{v}) = M(X,\mathbf{v})$. Towards a contradiction, suppose that $R^M$ is not a consistent correlation. Since $R^M$ is not a consistent correlation, there exist $X, X', \mathbf{v}, \mathbf{v}', \mathbf{w}, \mathbf{w}'$ such that $(X, [\![\mathbf{v}]\!], [\![\mathbf{w}]\!]) R^{M^n} (X', [\![\mathbf{v}']\!], [\![\mathbf{w}']\!])$, and

$$\exists \eta \in \Theta_{R^{M^n}}, \delta : [\![\varphi_X]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}]\!]/\mathbf{d}] \neq [\![\varphi'_X]\!]\eta\delta[[\![\mathbf{v}']\!]/\mathbf{c}, [\![\mathbf{w}']\!]/\mathbf{d}].$$

By definition of $R^{M^n}$, $X = X'$, and $[\![\mathbf{v}]\!] = [\![\mathbf{v}']\!]$, hence this is equivalent to

$$\exists \eta \in \Theta_{R^{M^n}}, \delta : [\![\varphi_X]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}]\!]/\mathbf{d}] \neq [\![\varphi_X]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}']\!]/\mathbf{d}]. \tag{6.3}$$

Let $\eta$ and $\delta$ be such, and let $\delta_1 = \delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}]\!]/\mathbf{d}]$ and $\delta_2 = \delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}']\!]/\mathbf{d}]$. Define $\varphi'_X \triangleq \mathsf{simplify}(\varphi_X[\mathbf{c} := \mathbf{v}])$. Since the values in $\mathbf{v}$ are closed, and from the definition of $\mathsf{simplify}$, we find that $[\![\varphi_X]\!]\eta\delta_1 = [\![\varphi'_X]\!]\eta\delta_1$, and likewise for $\delta_2$. Therefore, we know that

$$[\![\varphi'_X]\!]\eta\delta_1 \neq [\![\varphi'_X]\!]\eta\delta_2. \tag{6.4}$$

Observe that for all $\mathbf{d}[k] \in M$, $[\![\mathbf{w}[k]]\!] = [\![\mathbf{w}'[k]]\!]$ by definition of $R^M$. Every predicate variable instance that might change the solution of $\varphi'_X$ is a neighbour of $(X,\mathbf{v})$ in the control flow graph, according to Lemma 6.33. Take an arbitrary predicate variable instance $\mathsf{PVI}(\varphi_X, i) = Y(\mathbf{e}, \mathbf{e}')$ in $\varphi'_X$. We first show that $[\![\mathbf{e}'[\ell]]\!]\delta_1 = [\![\mathbf{e}'[\ell]]\!]\delta_2$ for all $\ell$.

Observe that $[\![\mathbf{e}]\!]\delta_1 = [\![\mathbf{e}]\!]\delta_2$ since $\mathbf{e}$ are expressions substituted for control flow parameters, and hence are either constants, or the result of copying.

Furthermore, there is no unmarked parameter $\mathbf{d}[k]$ that can influence a marked parameter $\mathbf{d}[\ell]$ at location $(Y,\mathbf{u})$. If there is a $\mathbf{d}[\ell] \in M^n(Y,\mathbf{u})$ such that $\mathbf{d}[k] \in \mathsf{free}(\mathbf{e}'[\ell])$, and $\mathbf{d}[k] \notin M^n(X,\mathbf{v})$, then by definition of marking $\mathbf{d}[k] \in M^{n+1}(X,\mathbf{v})$, which contradicts the assumption that the marking is stable, so it follows that

$$[\![\mathbf{e}'[\ell]]\!]\delta_1 = [\![\mathbf{e}'[\ell]]\!]\delta_2 \text{ for all } \ell. \tag{6.5}$$

From (6.5), and since we have chosen the predicate variable instance arbitrarily, it follows that for all $1 \leq i \leq \mathsf{npred}(\varphi'_X)$, $[\![X(\mathbf{e},\mathbf{e}')]\!]\eta\delta_1 = [\![X(\mathbf{e},\mathbf{e}')]\!]\eta\delta_2$. Together with (6.4), according to Lemma 6.38, this implies that there is some $d \in \mathsf{sig}(\varphi'_X)$ such that $\delta_1(d) \neq \delta_2(d)$. From the definition of $M^0$, however, it follows that $d$ must be marked in $M^0$, and hence also in $M^n$. According to the definition of $R^{M^n}$ it then is the case that $\delta_1(d) = \delta_2(d)$, which is a contradiction. Since also in this case we derive a contradiction, the original assumption that $R^M$ is not a consistent correlation does not hold, and we conclude that $R^M$ is a consistent correlation. □

A data parameter $d$ that is irrelevant at location $(X,\mathbf{v})$ can be assigned a default, fixed, value $\mathsf{init}(d)$ in any predicate variable instance $\mathsf{PVI}(\varphi_Y, i)$ in any equation $Y \in \mathsf{bnd}(\mathscr{E})$ for which $\mathsf{pred}(\varphi_Y, i) = X$ and for which the control flow parameters have value $\mathbf{v}$. This is exactly what the function $\mathsf{Reset}$, defined below, achieves.

**Definition 6.40.** Let $\mathscr{E}$ be a PBES, let $(V, \to)$ be its control flow graph, with marking $M$. Resetting a PBES is inductively defined on the structure of $\mathscr{E}$.

$$\begin{array}{lcl} \mathsf{Reset}_M(\epsilon) & \triangleq & \epsilon \\ \mathsf{Reset}_M(\sigma X(\mathbf{c}\colon \mathbf{C}, \mathbf{d}\colon \mathbf{D}) = \varphi)\mathscr{E}') & \triangleq & (\sigma \bar{X}(\mathbf{c}\colon \mathbf{C}, \mathbf{d}\colon \mathbf{D}) = \mathsf{Reset}_M(\varphi))\mathsf{Reset}_M(\mathscr{E}') \end{array}$$

Resetting for formulae is defined inductively as follows:

$$\begin{array}{lcl} \mathsf{Reset}_M(b) & \triangleq & b \\ \mathsf{Reset}_M(\varphi \wedge \psi) & \triangleq & \mathsf{Reset}_M(\varphi) \wedge \mathsf{Reset}_M(\psi) \\ \mathsf{Reset}_M(\varphi \vee \psi) & \triangleq & \mathsf{Reset}_M(\varphi) \vee \mathsf{Reset}_M(\psi) \\ \mathsf{Reset}_M(\forall d\colon D.\varphi) & \triangleq & \forall d\colon D.\mathsf{Reset}_M(\varphi) \\ \mathsf{Reset}_M(\exists d\colon D.\varphi) & \triangleq & \exists d\colon D.\mathsf{Reset}_M(\varphi) \\ \mathsf{Reset}_M(X(\mathbf{e}, \mathbf{e}')) & \triangleq & \bigwedge_{\mathbf{v} \in \mathsf{values}(\mathbf{c})} (\mathbf{e} = \mathbf{v} \implies \bar{X}(\mathbf{v}, \mathsf{Reset}^M_{(X,\mathbf{v})}(\mathbf{e}'))) \end{array}$$

With $\mathbf{e} = \mathbf{v}$ we denote that for all $i$, $\mathbf{e}[i] = \mathbf{v}[i]$. The function $\mathsf{Reset}^{(X,\mathbf{v})}_M(\mathbf{e}')$ is defined positionally as follows:

$$\mathsf{Reset}^{(X,\mathbf{v})}_M(\mathbf{e}')[i] = \begin{cases} \mathbf{e}'[i] & \text{if } \mathbf{d}[i] \in M(X, \mathbf{v}) \\ \mathsf{init}(\mathbf{d}[i]) & \text{otherwise.} \end{cases}$$

*Remark* 6.41. We can reduce the number of equivalence we introduce in resetting a recurrence. This effectively reduces the guard as follows.

Let $X \in \mathsf{bnd}(\mathscr{E})$, such that $Y(\mathbf{e}, \mathbf{e}') = \mathsf{PVI}(\varphi_X, i)$, and let $I = \{j \mid \mathsf{dest}(X, i, j) = \bot\}$ denote the indices of the control flow parameters for which the destination is undefined.

Define $\mathbf{c}' = \mathbf{c}[i_1], \ldots \mathbf{c}[i_n]$ for $i_n \in I$, and $\mathbf{f} = \mathbf{e}[i_1], \ldots \mathbf{e}[i_n]$ to be the vectors of control flow parameters for which the destination is undefined, and the values that are assigned to them in predicate variable instance $i$. Observe that these are the only control flow parameters that we need to constrain in the guard while resetting.

We can redefine $\mathsf{Reset}_M(X(\mathbf{e}, \mathbf{e}'))$ as follows.

$$\mathsf{Reset}_{M_{loc}}(X(\mathbf{e}, \mathbf{e}')) \triangleq \bigwedge_{\mathbf{v}' \in \mathsf{values}(\mathbf{c}')} (\mathbf{f} = \mathbf{v}' \implies \bar{X}(\mathbf{v}, \mathsf{Reset}^{M_{loc}}_{(X,\mathbf{v})}(\mathbf{e}'))).$$

In this definition $\mathbf{v}$ is defined positionally as

$$\mathbf{v}[j] = \begin{cases} \mathbf{v}'[j] & \text{if } j \in I \\ \mathsf{dest}(X, i, j) & \text{otherwise} \end{cases}$$

Resetting irrelevant parameters preserves the solution of the PBES. We formalise this in Theorem 6.46 below. Our proof is based on consistent correlation. We first define the relation $R^{\mathsf{Reset}}$, and we show that this is indeed a consistent correlation. Soundness then follows from Theorem 6.7. Note that $R^{\mathsf{Reset}}$ uses the relation $R^M$ from Definition 6.37 to relate predicate variable instances of the original equation system. The latter is used in the proof of Lemma 6.44.

**Definition 6.42.** Let $R^{\mathsf{Reset}}$ be the relation defined as follows.

$$\begin{cases} X([\![\mathbf{v}]\!],[\![\mathbf{w}]\!])R^{\mathsf{Reset}}\bar{X}([\![\mathbf{v}]\!],[\![\mathsf{Reset}_M^{(X,\mathbf{v})}(\mathbf{w})]\!]) & \\ X([\![\mathbf{v}]\!],[\![\mathbf{w}]\!])R^{\mathsf{Reset}}X([\![\mathbf{v}]\!],[\![\mathbf{w}']\!]) & \text{if } X([\![\mathbf{v}]\!],[\![\mathbf{w}]\!])R^{M}X([\![\mathbf{v}]\!],[\![\mathbf{w}']\!]) \end{cases}$$

Towards showing that $R^{\mathsf{Reset}}$ is a consistent correlation, we first show that we can unfold the values of the control flow parameters in every predicate variable instance, by duplicating the predicate variable instance, and substituting the values of the CFPs.

**Lemma 6.43.** *Let $\eta$ and $\delta$ be environments, and $X \in \mathsf{bnd}(\mathscr{E})$, then for all $i \leq \mathsf{npred}(\varphi_X)$, such that $\mathsf{PVI}(\varphi_X, i) = Y(\mathbf{e},\mathbf{e}')$,*

$$[\![Y(\mathbf{e},\mathbf{e}'))]\!]\eta\delta = [\![\bigwedge_{\mathbf{v}\in \mathsf{values}(\mathbf{c})} (\mathbf{e}=\mathbf{v} \implies Y(\mathbf{v},\mathbf{e}')]\!]\eta\delta$$

*Proof.* Straightforward; observe that $\mathbf{e}=\mathbf{v}$ for exactly one $\mathbf{v}\in\mathsf{values}(\mathbf{c})$, using that $\mathbf{v}$ is closed. □

Next we are establishing that resetting irrelevant parameters is sound, *i.e.*, it preserves the solution of the PBES. We do this by showing that $R^{\mathsf{Reset}}$ is a consistent correlation. We first show that resetting a predicate variable instance in an $R^{\mathsf{Reset}}$-correlating environment and a given data environment is sound.

**Lemma 6.44.** *Let $\mathscr{E}$ be a PBES, let $(V,\to)$ be its CFG, with marking $M$ such that $R^M$ is a consistent correlation, then*

$$\forall \eta \in \Theta_{R^{\mathsf{Reset}}}, \delta : [\![Y(\mathbf{e},\mathbf{e}')]\!]\eta\delta = [\![\mathsf{Reset}_M(Y(\mathbf{e},\mathbf{e}'))]\!]\eta\delta$$

*Proof.* Let $\eta \in \Theta_{R^{\mathsf{Reset}}}$, and $\delta$ be arbitrary. We derive this as follows.

$$\begin{array}{lll}
 & [\![\mathsf{Reset}_M(Y(\mathbf{e},\mathbf{e}')))]\!]\eta\delta & \\
= & \{\text{Definition 6.40}\} & \\
 & [\![\bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}(\mathbf{e}=\mathbf{v} \implies \bar{Y}(\mathbf{v},\mathsf{Reset}_M^{(Y,\mathbf{v})}(\mathbf{e}')))]\!]\eta\delta & \\
=^{\dagger} & \bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}([\![\mathbf{e}]\!]\delta=[\![\mathbf{v}]\!] \implies [\![\bar{Y}(\mathbf{v},\mathsf{Reset}_M^{(Y,\mathbf{v})}(\mathbf{e}')]\!]\eta\delta)) & \\
=^{\dagger} & \bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}([\![\mathbf{e}]\!]\delta=[\![\mathbf{v}]\!] \implies \eta(\bar{Y})([\![\mathbf{v}]\!]\delta,[\![\mathsf{Reset}_M^{(Y,\mathbf{v})}(\mathbf{e}')]\!]\delta)) & \\
= & \{\eta\in\Theta_{R^{\mathsf{Reset}}}\} & \\
 & \bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}([\![\mathbf{e}]\!]\delta=[\![\mathbf{v}]\!] \implies \eta(Y)([\![\mathbf{v}]\!]\delta,[\![\mathbf{e}']\!]\delta))) & \\
=^{\dagger} & \bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}([\![\mathbf{e}]\!]\delta=[\![\mathbf{v}]\!] \implies [\![Y(\mathbf{v},\mathbf{e}')]\!]\eta\delta)) & \\
=^{\dagger} & [\![\bigwedge_{\mathbf{v}\in\mathsf{CFL}(Y)}(\mathbf{e}=\mathbf{v} \implies Y(\mathbf{v},\mathbf{e}'))]\!]\eta\delta & \\
= & \{\text{Lemma 6.43}\} & \\
 & [\![Y(\mathbf{e},\mathbf{e}'))]\!]\eta\delta &
\end{array}$$

Here at $^{\dagger}$ we have used the semantics. □

By extending this result to the right-hand sides of equations, we can prove that $R^{\mathsf{Reset}}$ is a consistent correlation.

**Proposition 6.45.** *Let $\mathscr{E}$ be a PBES, and let $(V, \rightarrow)$ be a CFG, with marking $M$ such that $R^M$ is a consistent correlation. Let $X \in \mathsf{bnd}(\mathscr{E})$, with $\mathbf{v} \in CFL(X)$, then for all $\mathbf{w}$, and for all $\eta \in \theta_{R^{\mathsf{Reset}}}$ and $\delta$*

$$[\![\varphi_X]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}]\!]/\mathbf{d}] = [\![\mathsf{Reset}_M(\varphi_X)]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathsf{Reset}_M^{(X,\mathbf{v})}(\mathbf{w})]\!]/\mathbf{d}]$$

*Proof.* Let $\eta$ and $\delta$ be arbitrary, and define $\delta_r \stackrel{\Delta}{=} \delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathsf{Reset}_M^{(X,\mathbf{v})}(\mathbf{w})]\!]/\mathbf{d}]$. We first prove that

$$[\![\varphi_X]\!]\eta\delta_r = [\![\mathsf{Reset}_M(\varphi_X)]\!]\eta\delta_r \tag{6.6}$$

We proceed by induction on $\varphi_X$.

- $\varphi_X = b$. Since $\mathsf{Reset}_M(b) = b$ this follows immediately.
- $\varphi_X = Y(\mathbf{e})$. This follows immediately from Lemma 6.44.
- $\varphi_X = \forall y\colon D.\varphi$. We derive that $[\![\forall y\colon D.\varphi]\!]\eta\delta_r = \forall v \in \mathbb{D}.[\![\varphi]\!]\eta\delta_r[v/y]$. According to the induction hypothesis, and since we applied only a dummy transformation on $y$, we find that $[\![\varphi]\!]\eta\delta_r[v/y] = [\![\mathsf{Reset}_M(\varphi)]\!]\eta\delta_r[v/y]$, hence $[\![\forall y\colon D.\varphi]\!]\eta\delta_r = [\![\mathsf{Reset}_M(\forall y\colon D.\varphi)]\!]\eta\delta_r$.
- $\varphi_X = \exists y\colon D.\varphi$. Analogous to the previous case.
- $\varphi_X = \varphi_1 \wedge \varphi_2$. We derive that $[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta_r = [\![\varphi_1]\!]\eta\delta_r \wedge [\![\varphi_2]\!]\eta\delta_r$. If we apply the induction hypothesis twice we get $[\![\varphi_1 \wedge \varphi_2]\!]\eta\delta_r = [\![\mathsf{Reset}_M(\varphi_1)]\!]\eta\delta_r \wedge [\![\mathsf{Reset}_M(\varphi_2)]\!]\eta\delta_r$. Applying the semantics, and the definition of Reset we find this is equal to $[\![\mathsf{Reset}_M(\varphi_1 \wedge \varphi_2)]\!]\eta\delta_r$.
- $\varphi_X = \varphi_1 \vee \varphi_2$. Analogous to the previous case.

Hence we find that $[\![\mathsf{Reset}_M(\varphi_X)]\!]\eta\delta_r = [\![\varphi_X]\!]\eta\delta_r$. It now follows immediately from the observation that $R^M$ is a consistent correlation, and Definition 6.40, that $[\![\varphi_X]\!]\eta\delta_r = [\![\varphi_X]\!]\eta\delta[[\![\mathbf{v}]\!]/\mathbf{c}, [\![\mathbf{w}]\!]/\mathbf{d}]$. Our result follows by transitivity of $=$. □

The theory of consistent correlations now gives an immediate proof of soundness of resetting irrelevant parameters, which is formalised by the following theorem.

**Theorem 6.46.** *Let $\mathscr{E}$ be a PBES, with control flow graph $(V, \rightarrow)$ and marking $M$. For all $X$, $\mathbf{v}$ and $\mathbf{w}$:*

$$[\![\mathscr{E}]\!](X([\![\mathbf{v}]\!], [\![\mathbf{w}]\!])) = [\![\mathsf{Reset}_M(\mathscr{E})]\!](\bar{X}([\![\mathbf{v}]\!], [\![\mathbf{w}]\!])).$$

*Proof.* Relation $R^{\mathsf{Reset}}$ is a consistent correlation, as witnessed by Proposition 6.45. From Theorem 6.7 the result now follows immediately. □

The effect of resetting is that equations $\sigma X(\mathbf{d}\colon \mathbf{D}) = \varphi_X$ with different instances $\mathbf{v}$ for their formal parameters $\mathbf{d}$ may become more 'alike' after resetting, resulting in a potential reduction of the underlying Boolean equation system. This is nicely illustrated by applying the reset function on our running example.

**Example 6.47.** Applying Reset using the marking from Example 6.36 on the PBES of Example 6.8 results in the PBES below. Note that we have simplified the PBES slightly by removing the redundant conditions introduced by Reset, and by removing quantifiers that quantified over unused variables.

$$
\begin{aligned}
&\nu\bar{X}(s:N,d:D) = (\forall e:D.s=1 \implies \bar{Y}(2,e,e)) \wedge (s=1 \implies \bar{X}(2,d_1))\\
&\quad \wedge (s=2 \implies \bar{X}(3,d_1)) \wedge (s=2 \implies \bar{X}(4,d_1)) \wedge (s=3 \implies \bar{X}(1,d_1))\\
&\quad \wedge (s=4 \implies \bar{X}(1,d_1))\\
&\nu\bar{Y}(s:N,d,v:D) = (s=4 \implies d=v) \wedge (s=2 \implies \bar{Y}(3,d_1,d_1))\\
&\quad \wedge (s=2 \implies \bar{Y}(4,d,v)) \wedge (s=3 \implies \bar{Y}(1,d_1,d_1)) \wedge (s=4 \implies \bar{Y}(1,d_1,d_1))\\
&\textbf{init } \bar{X}(1,d_1);
\end{aligned}
$$

Note that $Y(2,d,v)$ depended on $Y(3,d,v)$; on the other hand, $\bar{Y}(2,d,v)$ depends on $\bar{Y}(3,d_1,d_1)$. In a way, the equations for $\bar{Y}(2,d,v)$ are more alike than those for $Y(2,d,v)$. This resemblance is also reflected in the size reduction after instantiation: for $|D| = 8$, the BES underlying the original PBES has 71 equations; the BES underlying the above PBES has 22 equations only.

## 6.4 Optimisation

The size of a CFG can grow exponentially in the number of control flow parameters in the worst case, hampering the efficiency of the static analysis. To counter this, we define a notion of local control flow. Then, we tailor the data flow analysis to this new situation to partly counter the loss of information that comes with these new control flow graphs.

We first illustrate that, indeed, the analysis as conducted in the previous section can suffer from an exponential blow-up. Typical scenarios that lead to such a blow-up are model checking problems of systems that consist of $N$ similar parallel components. In a worst-case situation, all the control flow parameters in the resulting PBES evolve orthogonally.

**Example 6.48.** Consider a system consisting of $N$ lossy buffers (from Example 6.8) running in parallel, not synchronising. Suppose we encode the deadlock freedom model checking problem, expressed by $\nu X.[\mathsf{true}]X \wedge \langle \mathsf{true} \rangle \mathsf{true}$ as a PBES. In the resulting PBES (omitted due to its size), consisting of one equation only, each independent lossy buffer can have a control flow parameter that can take on 4 different values. Then, depending on the choice of the set of control flow parameters, the resulting global control flow graph can have up-to $4^N$ locations.

Drawing inspiration from [PT09], we next define a variant of the control flow graphs of Section 6.2 that do not suffer from this blow-up. This permits trading of the power of the subsequent data flow analysis and the computational complexity. The terminology for the concepts we use in the remainder of this section and the next is borrowed largely from [PT09].

From hereon, assume that $\mathscr{E}$ is a fixed PBES, $(\mathsf{source}, \mathsf{dest}, \mathsf{copy})$ is a unicity constraint, and the vector $\mathbf{c}$ is a vector of control flow parameters. The local control flow

graph, which we define next, is a collection of unconnected graphs. Each connected subgraph represents a control flow parameter, and, in particular, the values it can assume, the equation in which it is considered, and which predicate variable instances it governs.

**Definition 6.49.** The *local* control flow graph is a graph $(V^{loc}, \underset{loc}{\longrightarrow})$ with:

- $V^{loc} = \{(X, n, v) \mid X \in \mathsf{bnd}(\mathscr{E}) \wedge n \leq |\mathbf{c}| \wedge v \in \mathsf{values}(\mathbf{c}[n])\}$, and
- $\underset{loc}{\longrightarrow} \subseteq V^{loc} \times \mathbb{N} \times V^{loc}$ is the least relation such that $(X, n, v) \xrightarrow[loc]{i} (\mathsf{pred}(\varphi_X, i), n, w)$ if:
  1. $\mathsf{source}(X, i, n) = v$ and $\mathsf{dest}(X, i, n) = w$, or
  2. $\mathsf{source}(X, i, n) = \bot$, $\mathsf{pred}(\varphi_X, i) \neq X$ and $\mathsf{dest}(X, i, n) = w$, or
  3. $\mathsf{source}(X, i, n) = \bot$, $\mathsf{pred}(\varphi_X, i) \neq X$ and $\mathsf{copy}(X, i, n) = n$ and $v = w$.

Note that the size of a local control flow graph is linear in the number of equations, the number of control flow parameters and the size of the set of values each individual control flow parameter can assume. In particular, the size of the local control flow graph for Example 6.48 would be $\mathcal{O}(4N)$ instead of $\mathcal{O}(4^N)$.

Observe that for the local control flow graph, the structural property that is similar to the one in Lemma 6.33 does not hold due to the absence of self-loops through copied control flow parameters in the local control flow graphs. The local control flow graph does contain self-loops in case both source and destination are defined.

The information we can obtain from a location in a local control flow graph is limited: it only tells the value of the control flow parameter it represents. Yet, the unicity constraint underlying the control flow parameter can be used to determine which predicate variable instances are potentially enabled in a location of the local control flow graph for a fixed control flow parameter; in this case, we say that the predicate variable instance is *ruled* by this control flow parameter. From hereon, we assume that $X \in \mathsf{bnd}(\mathscr{E})$ is an arbitrary bound predicate variable in the PBES $\mathscr{E}$ unless stated otherwise.

**Definition 6.50.** Control flow parameter $\mathbf{c}[j]$ *rules* $\mathsf{PVI}(\varphi_X, i)$ whenever there is a value $v$ for which $(X, j, v) \xrightarrow[loc]{i}$.

The predicate variable instances that are potentially enabled by a control flow parameter form a *cluster* of predicate variable instances. These clusters will be used to guide us in our data flow analysis. We first identify which data parameters are potentially used in the scope of a predicate variable instance.

**Definition 6.51.** Variable $d$ is *used* for $\mathsf{PVI}(\varphi_X, i)$ if $d \in \mathsf{free}(\mathsf{guard}^i(\varphi_X))$.

Parameters whose own values are potentially modified through a recursion are identified as *changed*. Observe that this only makes sense for self-recursions.

**Definition 6.52.** Parameter $\mathbf{d}[j] \in \mathsf{par}(X)$ is *changed* for $\mathsf{pred}(\varphi_X, i)$ if $X = \mathsf{pred}(\varphi_X, i)$ and $\mathbf{d}[j] \neq \mathsf{data}(\varphi_X, i)[j]$.

We can now formalise which data parameters have their data flow (*i.e.*, their use and their changes) entirely subsumed by such a cluster. If this is the case, we say that such a data parameter *belongs to* the cluster.

**Definition 6.53.** Let $c$ be a control flow parameter and let $d \in \mathsf{par}(X) \cap \mathscr{D}^{\mathrm{DP}}$ be a data parameter. We say that *$d$ belongs to $c$* if either:

1. $d$ is neither used nor changed for all $\mathsf{PVI}(\varphi_X, i)$ ruled by $c$, or
2. both the following hold:
   - whenever $d$ is used in $\mathsf{PVI}(\varphi_X, i)$, $c$ rules $\mathsf{PVI}(\varphi_X, i)$, and
   - whenever $d$ is changed for $\mathsf{PVI}(\varphi_X, i)$, $c$ rules $\mathsf{PVI}(\varphi_X, i)$.

The set of data parameters that belong to $c$ is denoted by $\mathsf{belongs}(c)$.

For ease of reasoning, we continue to work under the following assumption.

**Assumption 6.54.** Each right-hand side predicate formula in a PBES contains at least one predicate variable instance and each data parameter in an equation belongs to at least one CFP; CFPs belong to no parameter.

Observe that this assumption imposes no restrictions: equations $\sigma X(\mathbf{d}\colon \mathbf{D}) = \varphi_X$ where $\varphi_X$ contains no predicate variable instances, can be strengthened to $\varphi_X \wedge X(\mathbf{d})$ in case $\sigma = \nu$ and weakened to $\varphi_X \vee X(\mathbf{d})$ otherwise, without affecting the solution to $X$ or any of the other equations in an equation system. By adding a dummy parameter $b$ of sort $B$ to every equation $\sigma X(\mathbf{d}\colon \mathbf{D}) = \varphi_X$, initialising it to $\mathsf{true}$, strengthening each $\varphi_X$ to $\varphi_X \wedge b = \mathsf{true}$, and never changing $b$ in predicate variable instances, we effectively turn $b$ into a control flow parameter to which each data parameter can belong.

We identify relevant parameters in the local control flow graph in a way that is similar to how it is done in Section 6.3. First, in a control flow location $(X, n, v)$, those parameters that belong to control flow parameter $\mathbf{c}[n]$ are marked that may be significant in $\varphi_X[\mathbf{c}[n] := v]$. Then, additional data parameters are identified as being relevant by determining whether they can (indirectly) affect a parameter that was already determined to be significant. For the soundness of the analysis, care must be taken that this also works in case a data parameter $d$ that belongs to one control flow parameter affects a data parameter $d'$ that belongs to another: in case the latter is already marked relevant, this requires that $d$ is marked relevant too.

**Definition 6.55.** Let $(V^{loc}, \underset{loc}{\longrightarrow})$ be a local control flow graph for PBES $\mathscr{E}$. We define local marking $M_{loc}\colon V^{loc} \to \mathbb{P}(\mathscr{D}^{\mathrm{DP}})$ inductively as follows:

$$
\begin{aligned}
M^0_{loc}(X,n,v) &= \{d \in \mathsf{belongs}(\mathbf{c}[n]) \mid d \in \mathsf{sig}(\mathsf{simplify}(\varphi_X[\mathbf{c}[n] := v]))\}\\
M^{k+1}_{loc}(X,n,v) &= M^k_{loc}(X,n,v)\\
&\quad \cup \{d \in \mathsf{belongs}(\mathbf{c}[n]) \mid \exists w\ \exists \mathbf{d}^Y[\ell] \in M^k_{loc}(Y,n,w)\\
&\qquad (X,n,v) \underset{loc}{\xrightarrow{i}} (Y,n,w) \wedge d \text{ affects } \mathsf{data}(\varphi_X,i)[\ell]\}\\
&\quad \cup \{d \in \mathsf{belongs}(\mathbf{c}[n]) \mid \exists m,w\ \exists \mathbf{d}^Y[\ell] \in M^k_{loc}(Y,m,w)\\
&\qquad \mathbf{d}[\ell] \notin \mathsf{belongs}(\mathbf{c}[n]) \wedge Y = \mathsf{pred}(\varphi_X,i) \wedge\\
&\qquad d \text{ affects } \mathsf{data}(\varphi_X,i)[\ell]\}\\
M_{loc}(X,n,v) &= \textstyle\bigcup_{k\in\mathbb{N}} M^k_{loc}(X,n,v)
\end{aligned}
$$

The local marking can again be used to reset data parameters using function Reset. We first define the induced marking for (global) control flow locations.

**Definition 6.56.** The induced marking $M_{loc}(X, \mathbf{v})$ is defined as $d \in M_{loc}(X, \mathbf{v})$ iff $\forall j : d \in \mathsf{belongs}(\mathbf{c}[j]) \implies d \in M_{loc}(X, j, \mathbf{v}[j])$.

This induced marking overapproximates the marking computed in Section 6.3, as is shown by the following lemma.

**Lemma 6.57.** *Let $\mathscr{E}$ be a PBES with global CFG $(V, \longrightarrow)$ with marking $M$, and local CFG $(V^{loc}, \underset{loc}{\longrightarrow})$ with marking $M_{loc}$. For all natural numbers $n$ it holds that $\forall (X, \mathbf{v}) \in V, \forall d \in M^n(X, \mathbf{v}) : (\forall j : d \in \mathsf{belongs}(\mathbf{c}[j]) \implies d \in M^n_{loc}(X, j, \mathbf{v}[j]))$.*

*Proof.* We proceed by induction on $n$.

- $n = 0$. Let $(X, \mathbf{v})$ and $d \in M^0(X, \mathbf{v})$ be arbitrary. We need to show that $\forall j : d \in \mathsf{belongs}(\mathbf{c}[j]) \implies d \in M^0_{loc}(X, j, \mathbf{v}[j])$.

  Let $j$ be arbitrary such that $d \in \mathsf{belongs}(\mathbf{c}[j])$. Since $d \in M^0(X, \mathbf{v})$, by definition $d \in \mathsf{sig}(\mathsf{simplify}(\varphi_X[\mathbf{c} := \mathbf{v}]))$, hence also $d \in \mathsf{sig}(\mathsf{simplify}(\varphi_X[\mathbf{c}[j] := \mathbf{v}][j]))$. Combined with the assumption that $d \in \mathsf{belongs}(\mathbf{c}[j])$, this gives us $d \in M^0_{loc}(X, j, \mathbf{v}[j])$ according to Definition 6.49.

- $n = m + 1$. As induction hypothesis assume $\forall (X, \mathbf{v}) \in V : \forall d : d \in M^m(X, \mathbf{v}) \implies (\forall j : d \in \mathsf{belongs}(\mathbf{c}[j]) \implies d \in M^m_{loc}(X, j, \mathbf{v}[j]))$. Now let $(X, \mathbf{v})$ be arbitrary, and let $d \in M^{m+1}(X, \mathbf{v})$. Also let $j$ be arbitrary, and assume that $d \in \mathsf{belongs}(\mathbf{c}[j])$.

  We need to show that $d \in M^{m+1}_{loc}(X, j, \mathbf{v}[j])$. We proceed by distinction on the cases of Definition 6.35. If $d \in M^m(X, \mathbf{v})$ the result follows immediately from the induction hypothesis.

  Now suppose there is an $i \leq \mathsf{npred}(\varphi_X)$ such that $(X, \mathbf{v}) \xrightarrow{i} (\mathsf{pred}(\varphi_X, i), \mathbf{w})$, and there is some $\mathbf{d}[\ell] \in M^m(\mathsf{pred}(\varphi_X, i), \mathbf{w})$ with $d \in \mathsf{free}(\mathsf{data}(\varphi_X, i)[\ell])$

  Let $i$ and $\mathbf{d}[\ell]$ be such.

  According to the induction hypothesis, $\forall k : \mathbf{d}[\ell] \in \mathsf{belongs}(\mathbf{c}[k]) \implies \mathbf{d}[\ell] \in M^m_{loc}(\mathsf{pred}(\varphi_X, i), k, \mathbf{w}[k])$.

  - $\mathbf{d}[\ell]$ belongs to $\mathbf{c}[j]$. According to the induction hypothesis we have $\mathbf{d}[\ell] \in M^m_{loc}(\mathsf{pred}(\varphi_X, i), j, \mathbf{w}[j])$. We have $d \in \mathsf{free}(\mathsf{data}(\varphi_X, i)[\ell])$, so we only need to show that $(X, j, \mathbf{v}[j]) \xrightarrow[loc]{i} (\mathsf{pred}(\varphi_X, i), j, \mathbf{w}[j])$. We distinguish the cases for $j$ from Definition 6.31.
    - $\mathsf{source}(X, i, j) = \mathbf{v}[j]$ and $\mathsf{dest}(X, i, j) = \mathbf{w}[j]$, then according to Definition 6.49 the edge $(X, j, \mathbf{v}[j]) \xrightarrow[loc]{i} (\mathsf{pred}(\varphi_X, i), j, \mathbf{w}[k])$ also exists.
    - $\mathsf{source}(X, i, j) = \bot$, $\mathsf{copy}(X, i, j) = j$ and $\mathbf{v}[j] = \mathbf{w}[j]$. If $\mathsf{pred}(\varphi_X, i) \neq X$ the edge exists locally, and we are done. Now suppose that $\mathsf{pred}(\varphi_X, i) = X$. Then $\mathsf{PVI}(\varphi_X, i)$ is not ruled by $\mathbf{c}[j]$. Furthermore, $\mathbf{d}[\ell]$ is changed in $\mathsf{PVI}(\varphi_X, i)$, hence $\mathbf{d}[\ell]$ cannot belong to $\mathbf{c}[j]$, which is a contradiction.

  - $\mathsf{source}(X,i,j) = \bot$, $\mathsf{copy}(X,i,j) = \bot$ and $\mathsf{dest}(X,i,j) = \mathbf{w}[j]$. This is completely analogous to the previous case.
- $\mathbf{d}[\ell]$ does not belong to $\mathbf{c}[j]$. Recall that there must be some $\mathbf{c}[k]$ such that $\mathbf{d}[\ell]$ belongs to $\mathbf{c}[k]$, and by assumption now $\mathbf{d}[\ell]$ does not belong to $\mathbf{c}[j]$. Then according to Definition 6.55, $d$ is marked in $M_{loc}^{m+1}(X,j,\mathbf{v}[j])$, which is what we need to prove.

□

Note that the above lemma is stronger than what we rely on for proving correctness of the local analysis in order to facilitate the inductive proof. For correctness, the following corollary is sufficient.

**Corollary 6.58.** *Let, for given PBES $\mathscr{E}$, $(V,\longrightarrow)$ be a global control flow graph with marking $M$, and let $(V^{loc},\underset{loc}{\longrightarrow})$ be a local control flow graph with induced marking $M_{loc}$. Then $M(X,\mathbf{v}) \subseteq M_{loc}(X,\mathbf{v})$ for all $(X,\mathbf{v})$.*

The marking induced by the local analysis can again be used to reset data parameters without affecting the solution to the equation system.

**Theorem 6.59.** *Let $\mathscr{E}$ be a PBES, with local control flow graph $(V^{loc},\underset{loc}{\longrightarrow})$ and induced marking $M_{loc}$. Then for all $X$, $\mathbf{v}$ and $\mathbf{w}$:*

$$[\![\mathscr{E}]\!](X(\mathbf{v},\mathbf{w})) = [\![\mathsf{Reset}_{M_{loc}}(\mathscr{E})]\!](\bar{X}(\mathbf{v},\mathbf{w})).$$

*Proof.* Correctness of this theorem follows from Corollary 6.58 and Theorem 6.46. □

## 6.5 Case Studies

We implemented the techniques described in the previous sections in the context of the mCRL2 toolset [Cra+13], in the tool `pbesstategraph`. Here, we report on the tool's effectiveness in simplifying the PBESs originating from model checking problems and behavioural equivalence checking problems: we compare sizes of the BESs underlying the original PBESs to those underlying the PBESs obtained after running the tool `pbesparelm`, which implements the techniques from [OWW09], to those underlying the PBESs obtained after running our tool. The techniques implemented in `pbesparelm` effectively perform an analysis that removes parameters from an equation if they neither occur significantly, nor influence other parameters that in turn occur significantly.

Our cases are taken from the case studies presented in the previous chapter. We restrict ourselves to the cases in that chapter that use parameterised Boolean equation systems as an intermediate format. We present a representative selection of the results in Table 6.1. The full results are available on GitHub.[2] For the model checking problems, we consider the *Onebit* protocol, which is a sliding window protocol, and Hesselink's handshake register [Hes98]. Both protocols are parametric in the set of values that can be read and written. A selection of properties of varying complexity and varying nesting degree, expressed in the data-enhanced modal $\mu$-calculus are checked. For the behavioural

[2] https://github.com/jkeiren/pbesstategraph-experiments

equivalence checking problems, we consider a number of communication protocols such as the *Alternating Bit Protocol* (ABP), the *Concurrent Alternating Bit Protocol* (CABP), a two-place buffer (Buf) and the aforementioned Onebit protocol. Moreover, we compare an implementation of Hesselink's register to a specification of the protocol that is correct with respect to trace equivalence (but for which currently no PBES encoding exists) but not with respect to the two types of behavioural equivalence checking problems we consider here: branching bisimilarity and weak bisimilarity.

Our experiments reveal that the reductions achieved by our technique can lead to BESs that are about 90% smaller than those obtained after `pbesparelm`, see the model checking problems and equivalence checking problems for Hesselink's register. Compared to the sizes of the BESs underlying the original PBESs, the reductions can be immense.

## 6.6 Closing Remarks

In this chapter we described a static analysis technique inspired by [PT09], and applied to PBESs, that employs a notion of control flow to determine when data parameters become irrelevant. Using this information, the PBES can be simplified, leading to smaller underlying Boolean equation systems. Compared to existing techniques, our new static analysis technique can lead to greater reductions (up-to 90% in extreme cases), as also demonstrated in our case studies.

Several techniques described in this chapter can be used to enhance existing reduction techniques for PBESs. For instance, our notion of a *guard* of a predicate variable instance in a PBES can be put to use to cheaply improve on the heuristics for constant elimination [OWW09]. Moreover, we believe that our (re)construction of control flow graphs from PBESs can be used to automatically generate invariants for PBESs. The theory on invariants for PBESs is well-established [OW10], but still lacks tool support that can use invariants in automated solvers.

The correctness proofs of our static analysis techniques rely on the notion of consistent correlation. Consistent correlations generalise idempotence identifying bisimulation from Chapter 3. Our experimental results show that static analysis of PBESs allows for large reductions of the underlying Boolean equation systems. In the previous chapter, we have seen that weaker notions of equivalence allow for larger reductions of parity games. It is therefore interesting to explore generalisations of, especially, governed stuttering equivalence, to obtain equivalence notions for PBESs that are more general than consistent correlations. This can give rise to more advanced techniques for the *a priori* reduction in the context of verification using PBESs.

Table 6.1: Sizes of the BESs underlying the original PBESs, the PBESs reduced using `pbesparelm` and the PBESs reduced using `pbesstategraph`. The numbers reported reflect the number of equations generated using instantiation. Verdict $\surd$ means the outcome of the verification problem is true; $\times$ means the outcome is false.

| | | Original | `pbesparelm` | `pbesstategraph` | verdict |
|---|---|---|---|---|---|
| Model Checking Problems | | | | | |
| **No deadlock** | | | | | |
| *Onebit* | $\|D\| = 2$ | 81 921 | 11 409 | 9 089 | $\surd$ |
| | $\|D\| = 3$ | 289 297 | 11 409 | 9 089 | $\surd$ |
| | $\|D\| = 4$ | 742 401 | 11 409 | 9 089 | $\surd$ |
| *Hesselink* | $\|D\| = 2$ | 540 737 | 2 065 | 2 065 | $\surd$ |
| | $\|D\| = 3$ | 13 834 801 | 2 065 | 2 065 | $\surd$ |
| **No spontaneous generation of messages** | | | | | |
| *Onebit* | $\|D\| = 2$ | 185 089 | 30 593 | 22 145 | $\surd$ |
| | $\|D\| = 3$ | 1 278 433 | 57 553 | 39 169 | $\surd$ |
| | $\|D\| = 4$ | 5 588 481 | 92 289 | 60 161 | $\surd$ |
| **Messages that are read are inevitably sent** | | | | | |
| *Onebit* | $\|D\| = 2$ | 153 985 | 57 553 | 41 473 | $\times$ |
| | $\|D\| = 3$ | 579 745 | 115 489 | 78 817 | $\times$ |
| | $\|D\| = 4$ | 1 549 057 | 192 865 | 127 233 | $\times$ |
| **Messages can overtake one another** | | | | | |
| *Onebit* | $\|D\| = 2$ | 164 353 | 61 441 | 44 609 | $\times$ |
| | $\|D\| = 3$ | 638 065 | 127 153 | 88 225 | $\times$ |
| | $\|D\| = 4$ | 1 735 681 | 216 193 | 146 049 | $\times$ |
| **Values written to the register can be read** | | | | | |
| *Hesselink* | $\|D\| = 2$ | 1 093 761 | 1 081 345 | 89 089 | $\surd$ |
| | $\|D\| = 3$ | 27 876 961 | 27 656 641 | 561 169 | $\surd$ |
| Equivalence Checking Problems | | | | | |
| **Branching bisimulation equivalence** | | | | | |
| *ABP-CABP* | $\|D\| = 2$ | 31 265 | 31 265 | 30 225 | $\surd$ |
| | $\|D\| = 4$ | 73 665 | 73 665 | 69 681 | $\surd$ |
| *Buf-Onebit* | $\|D\| = 2$ | 844 033 | 706 561 | 511 554 | $\surd$ |
| | $\|D\| = 4$ | 8 754 689 | 5 939 201 | 3 707 138 | $\surd$ |
| *Hesselink I-S* | $\|D\| = 2$ | 21 062 529 | 21 062 529 | 1 499 714 | $\times$ |
| **Weak bisimulation equivalence** | | | | | |
| *ABP-CABP* | $\|D\| = 2$ | 50 713 | 49 617 | 47 481 | $\surd$ |
| | $\|D\| = 4$ | 117 337 | 113 361 | 106 089 | $\surd$ |
| *Buf-Onebit* | $\|D\| = 2$ | 966 897 | 706 033 | 552 226 | $\surd$ |
| | $\|D\| = 4$ | 9 868 225 | 5 869 505 | 3 862 402 | $\surd$ |
| *Hesselink I-S* | $\|D\| = 2$ | 29 868 273 | 28 579 137 | 2 067 650 | $\times$ |

# Chapter 7

# Conclusions

Parity games and parameterised Boolean equation systems are frameworks that are suitable for the verification of complex software systems. Their main advantage is that they combine a model of the system and the desired properties into a single framework. Solving the verification question hence requires the analysis of a single artefact. The drawback of the approach is the state space explosion that occurs when generating a parity game.

In this thesis we have investigated reduction techniques for parity games and parameterised Boolean equation systems, in which our main focus has been on those games and equation systems encoding model checking and equivalence checking problems.

In Chapter 3 we have provided a graph structure for Boolean equation systems with arbitrary right hand sides, and allowing the occurrence of free variables, answering the thus far unanswered question whether such a graph structure could be provided. In the same chapter, we have investigated the effects of normalisation, *i.e.* the transformation of a Boolean equation system into simple recursive form, in which the right hand side of an equation contains occurrences of at most one binary operator, *i.e.* conjunctions and disjunctions do not occur mixed in a single right hand side, and the equation system does not contain occurrences of $\mathsf{true}$ and $\mathsf{false}$. We have demonstrated that this transformation does not negatively impact the reducing capabilities of strong bisimulation reduction; in fact, it can lead to an arbitrarily larger reduction.

Strong bisimulation on structure graphs satisfies the nice property that bisimilar states in the state space lead to bisimilar states in the structure graph underlying the Boolean equation system encoding a model checking problem. To obtain this result, it is crucial that the translation of the model checking problem into a Boolean equation system preserves conjunctions and disjunctions. This is achieved using a harmless modification of Mader's translation. The resulting equation systems look awkward at times; they contain vertices of, *e.g.*, the form $f \vee f$, which we easily see can be simplified further. To achieve this, we have presented a weakening of strong bisimulation on structure graph that we dub idempotence identifying bisimulation.

The investigation of structure graphs suggests that the translation to restricted formats that has been commonly used in the literature is a useful one. In Chapter 4 we have therefore studied weaker equivalences for parity games. First we showed that stuttering

equivalence can be used to reduce parity games, and we weaken this equivalence further allowing vertices in a parity game that are owned by different players to be related under some circumstances.

The practical use of the equivalences that we have defined was investigated in Chapter 5. Here it was shown that, in general, reduction using our equivalences is able to reduce parity games by a large factor. In some cases it is even able to effectively solve parity games by reducing them to a single vertex. Unfortunately, in general the reductions do not prove to reduce the time required for model checking, although in general reducing and then solving a parity game is competitive with outright solving the original parity game. Note that no standard benchmark set for parity games is available. In this chapter we have presented a set of benchmarks for games that subsumes all parity game benchmarks in the literature. We have illustrated that the parity games that we present contain a nice mixture of games from different application domains, and of varying complexity.

In our final chapter, Chapter 6 we have improved the known static analysis techniques for parameterised Boolean equation systems by taking their control flow into account. Parameterised Boolean equation systems are, effectively, symbolic descriptions of Boolean equation systems, and their verification is undecidable. We have shown that using our control flow reconstruction, the size of the underlying equation system can be reduced by about 90% in some cases. The proof technique that we used to show the correctness of this static analysis technique is closely related to the equivalences that we presented in Chapter 3. In fact, Willemse showed [Wil10] that consistent correlations on Boolean equation systems correspond to idempotence identifying bisimulation.

**Future Work.** The work in this thesis serves as a basis for addressing the state space explosion problem in the verification of complex software systems using intermediate representations. To advance the practical verification of such systems, further research is required. In the rest of this chapter we discuss this future work.

In this thesis we have restricted ourselves to studying equivalences that only relate vertices if they have the same priority. Fritz and Wilke [FW06] studied preorders and equivalences that are able to relate vertices with different priorities that are incomparable to our equivalences. The drawback of these is their poor worst-case running time complexity. To enhance the understanding of the fundamental complexity of parity games, it could be investigated whether preorders corresponding to stuttering equivalence and governed stuttering equivalence can be developed that preserve or approximate the solution of parity games. Weak equivalences could also be developed that relate vertices with different priorities, but given our experiences in defining governed stuttering equivalence, proving its correctness, and preliminary explorations in this area, we expect that this proves extremely challenging.

We have shown that, in terms of size reductions, our equivalences are an effective tool in software verification. The drawback of the approach presented in Chapters 3 and 4 is that it still requires the explicit construction of a parity game, before it can be reduced. In Chapter 6 we showed that symbolic techniques are promising, and large reductions of the underlying parity game can be obtained *a priori*. The effort required in proving correctness of this transformation was greatly reduced by the coinductive nature of consistent correlations. Classical correctness proofs would require tedious and error-prone

proofs using transfinite induction. Consistent correlations, however, are a generalisation of the equivalences presented in Chapter 3, and hence do not compress sequences of equivalent vertices in the induced parity games. Our benchmarks indicate that reductions using governed stuttering equivalence lead to much smaller parity games than the reductions based on governed bisimulation. Techniques similar to consistent correlations, and corresponding static analysis for parameterised Boolean equation systems, are hence an interesting topic to explore. This should lead to more advanced techniques for the *a priori* reduction of parity games underlying parameterised Boolean equation systems.

We feel that the specific directions in which these techniques are developed should mainly be driven by the need observed when applying the theory to practical case studies. Interesting domains to focus on are, *e.g.*, the verification of distributed systems—in which often the degree of synchronisation between components is low— and the verification of low-level components in operating system kernels—in which parallel access to data structure is required, and in which locking is increasingly becoming problematic, hence introducing the need for more complex, wait-free data structures. In both areas intricate problems arise due the concurrency that is involved, giving rise to verification questions. Especially in distributed systems, the algorithms and protocols involved are becoming increasingly complex, and the number of nodes involved is growing rapidly, see *e.g.* the case study on verification of the detector controls software of the CMS detector of the large hadron collider at CERN [Hwo+13].

Currently, we foresee two candidates that might prove useful in these practical applications: confluence and cones and foci, which we discuss in more detail below. Both techniques are generalisations of existing techniques that were developed for labelled transition systems, along with their symbolic, process algebraic descriptions. In their original form, both were developed out of a clear practical need while doing case studies, and their generalisations to equation systems might prove effective as well.

In the generation of labelled transition systems from abstract, process algebraic specifications, the notion of *confluence* has proven to be an effective tool for reducing the size of the state space. Techniques have been developed to prove confluence based on a symbolic description of a state space, see *e.g.* [GS95; GS96; GP00; BP02]. Correctness of confluence reduction relies on branching bisimilarity of the underlying transition systems. We think that our notion of governed stuttering equivalence could be used to develop a notion of confluence for parity games, that in turn may lead to a definition of confluence reduction techniques for parameterised Boolean equation systems.

A related direction of research can be found in generalising the cones and foci approach [GS01; FP03; FPP06], that was originally developed for symbolic descriptions of processes to parameterised Boolean equation systems. It is expected that here the control flow graphs, and the notion of guards that we have developed in Chapter 6 prove as a helpful means for defining the required proof rules and analysing the equation systems. Note that the cones and foci method relies on a notion of invariants for symbolic descriptions of processes. A similar notion for parameterised Boolean equation systems has already been established [OW10].

# Appendix A

# Benchmarks

In this appendix we provide the formulae of the benchmarks that we described in Chapter 5. Specifically, we list the formulae for the model checking examples that we used in Chapters 5 and 6, and the formulae used for satisfiability and validity checking in Chapter 5.

The scripts used to generate the results, and the complete data of the experiments in Chapter 5 and Chapter 6 are available from their GitHub repositories.[1][2]

## A.1 Model Checking

All formulae used for model checking are denoted in the first order modal $\mu$-calculus, an mCRL2-native data extension of the modal $\mu$-calculus. The formulae assume that there is a data specification defining a non-empty sort $D$ of messages, and a set of parameterised actions that are present in the protocols.

### A.1.1 Communication Protocols

We verify a number of communication protocols. Here we give the generic versions of the properties that we verify. For specific protocols, the action names and the properties that apply may vary. We assume that the protocol receives messages via action $r$ and tries to send these to the other party. The other party can receive these via action $s$.

#### General Properties

- No deadlock:

  $$\nu X.[\mathsf{true}]X \wedge \langle \mathsf{true} \rangle \mathsf{true}$$

  Invariantly, over all reachable states at least one action is enabled.

[1] https://github.com/jkeiren/paritygame-generator
[2] https://github.com/jkeiren/pbesstategraph-experiments

- Invariantly, if a message can be read infinitely often it is read infinitely often:

$$\forall d : D.(\nu W.[\mathsf{true}]W \wedge \\ (\nu X.\mu Y.\nu Z.([r(d)]X \wedge ([r(d)]\mathsf{false} \vee [\overline{r(d)}]Y) \wedge [\overline{r(d)}]Z)))$$

- There is a path along which a message can be lost infinitely often:

$$\mu W\langle\mathsf{true}\rangle W \vee (\exists d : D.\langle r(d)\rangle(\nu X.\mu Y.(\langle c(e)\rangle X \vee \overline{\langle c(e) \vee s(d)\rangle}Y)))$$

  This formula assumes that losing a message is represented by the action $c(e)$.

- Along every path in which a message is read infinitely often, also a message is sent infinitely often:

$$\nu X.\mu Y.\nu Z.([\exists d : D.s(d)]X \wedge [\exists d : D.r(d)]Y \wedge [\overline{\exists d : D.r(d) \vee s(d)}]Z)$$

  That is, any sequence that contains infinitely many $r$ actions contains infinitely many $s$ actions.

- Message $d1$ can be received infinitely often:

$$\nu X.\mu Y.(\langle r(d1)\rangle X \vee \overline{\langle r(d1)\rangle}Y)$$

  From the initial state of the protocol, there is a path along which $d1$ is received infinitely often.

- All messages can be received infinitely often:

$$\forall d : D.\nu X.\mu Y.(\langle r(d)\rangle X \vee \overline{\langle r(d)\rangle}Y)$$

  This generalises the previous property to all messages.

- From any reachable state, there exists a path with infinitely many $\tau$-steps, and possibly some other steps [PW08].

$$\nu X.([\mathsf{true}]X \wedge \nu Z.\mu Y.(\langle\mathsf{true}\rangle Y \vee \langle\tau\rangle Z))$$

- The protocol does not duplicate messages:

$$\nu X.[\mathsf{true}]X \wedge \\ \forall d : D.[r(d)](\nu Y.[\overline{r(d) \vee s(d)}]Y \wedge [s(d)](\nu Z.[\overline{r(d)}]Z \wedge [s(d)]]\mathsf{false}))$$

  This property states that, after reading a message $d$, and subsequently sending the same $d$, it cannot be sent again, unless it has been read again.

- The protocol does not generate new messages:

$$\forall d : D.\nu X.[\overline{r(d)}]X \wedge [s(d)]\mathsf{false}$$

  From the initial state, no message can be sent unless a message has been read first.

- Messages that are read are inevitably sent:

$$\nu W.[\mathsf{true}]W \wedge (\forall d : D.([r(d)](\nu X.\mu Y.([s(d)]X \wedge [\overline{s(d)}]Y))))$$

- Messages that are read are inevitably sent along $i$-fair paths:

$$\nu X.[\mathsf{true}]X \wedge \forall d : D.([r(d)](\nu Y.\mu Z.([\overline{s(d) \vee i}]Z \wedge [i]Y)))$$

  This generalises the previous property to take fairness into account. The ABP, *e.g.*, non-deterministically (using an $i$ action) chooses whether to loose the message, or to forward it. This property only considers paths in which this choice is resolved fairly.

**Onebit Protocol**

For the onebit protocol we verify two additional properties. Note that the onebit protocol uses actions *ra* and *sb* instead of $r$ and $s$.

- Messages can be overtaken by other messages:

$$\begin{array}{l}\mu X.\langle\mathsf{true}\rangle X \vee \exists d : D.\langle ra(d)\rangle \mu Y. \\ \quad \Big( \langle \overline{sb(d)} \rangle Y \vee \exists d' : D.d \neq d' \wedge \langle ra(d')\rangle \mu Z. \\ \quad\quad (\langle \overline{sb(d)} \rangle Z \vee \langle sb(d')\rangle \mathsf{true}) \\ \quad \Big)\end{array}$$

  That is, there is a trace in which message $d$ is read, and is still in the protocol when another message $d'$ is read, which then is sent to the receiving party before message $d$.

- No spontaneous messages are generated:

$$\begin{array}{l}\nu X.[\overline{\exists d : D.ra(d)}]X \wedge \\ \quad \forall d' : D.[ra(d')]\nu Y(m_1 : D = d'). \\ \quad\quad \Big( [\overline{\exists d : D.ra(d) \vee sb(d)}]Y(m_1) \wedge \\ \quad\quad\quad \forall e : D.[sb(e)]((m_1 = e) \wedge X) \wedge \\ \quad\quad\quad \forall e' : D.[ra(e')]\nu Z(m_2 : D = e'). \\ \quad\quad\quad\quad \Big( [\overline{\exists d : D.ra(d) \vee sb(d)}]Z(m_2) \wedge \\ \quad\quad\quad\quad\quad \forall f : D.[sb(f)]((f = m_1) \wedge Y(m_2)) \\ \quad\quad\quad\quad \Big) \\ \quad\quad \Big)\end{array}$$

  Since the onebit protocol can contain two messages at a time, the formula states that only messages that are received can be subsequently sent again. This requires storing messages that are currently in the buffer using parameters $m_1$ and $m_2$.

**Bounded Retransmission Protocol**

The bounded retransmission protocol only allows a finite number of retransmissions, before finally giving up. The following property nicely shows the expressive power of the data-enhanced $\mu$-calculus.

$$
\begin{array}{l}
\nu X.[\mathsf{true}]X \wedge \forall l : \mathit{List}(D).\Big(l \neq [] \implies [r1(l)] \\
\quad \nu Y(n : \mathit{Nat} = 0, l' : \mathit{List}(D) = l).\Big( \\
\quad\quad [c9(\mathit{lost})]Y(n+1, l') \\
\quad\quad \wedge\ [\overline{c9(\mathit{lost}) \vee s1(I\_nok) \vee s1(I\_dk) \vee \exists d : D, i : \mathit{Ind}.s4(d,i)}]Y(n, l') \\
\quad\quad \wedge\ (l' \neq [] \implies \forall d : D.[\exists i : \mathit{Ind}.s4(d,i)](d = \mathit{head}(l') \wedge Y(0, \mathit{tail}(l')))) \\
\quad\quad \wedge\ ((n < \mathit{MAX} \wedge l' \neq []) \implies (\mu Z.\langle\mathsf{true}\rangle Z \vee \langle \exists i : \mathit{Ind}.s4(\mathit{head}(l'), i)\rangle\mathsf{true})) \\
\quad \Big) \\
\Big)
\end{array}
$$

This states that, invariantly, if a non-empty list of messages is received through $r1$, it is possible to send a message. The inner equation for $Y$ is parameterised by the number of times $n$ sending has been retried, and the list of messages $l'$ that still need to be send. If a message is lost, then the counter is incremented and the list is unchanged. For internal steps, the counter and the list remain unchanged. If there still is a message to be sent, then the only message that can be sent is the first message in $l'$, and afterwards the property again holds for the remaining messages. As long as the maximum number of retries has not been reached, and their still is a message to be sent, there indeed is a path to a state in which the first message in $l'$ can be sent.

### A.1.2 Cache Coherence Protocol

For the cache coherence protocol, we verify a number of properties from [Pan+07].

- Each region has at most one home node. $\nu X.[\mathsf{true}]X \wedge [c\_home]\mathsf{false}$ Note that here *c_home* is an action that is the result of communication in the model.
- If the system is stable, each region has no more than $n-1$ copies, where $n$ is the number of processors.

$$
\begin{array}{l}
\forall p : \mathit{ProcessId}.\neg\Big( \\
\quad \mu X.\langle\mathsf{true}\rangle X \vee \Big( \\
\quad\quad \langle c\_copy\rangle\mathsf{true} \wedge \\
\quad\quad \langle \mathit{lockempty}(p)\rangle\mathsf{true} \wedge \\
\quad\quad \langle \mathit{homequeueempty}(p)\rangle\mathsf{true} \wedge \\
\quad\quad \langle \mathit{remotequeueempty}(p)\rangle\mathsf{true} \\
\quad\quad \Big) \\
\quad \Big)
\end{array}
$$

This property works for a model with two processors, checking that there is no state in which the actions *c_copy*, *lockempty*, *homequeueempty* and *remotequeueempty* are enabled simultaneously.

- Every thread eventually finishes writing to a region:

$$\begin{array}{l}\forall t : \mathit{ThreadId}. \\ \quad \nu X.[\mathsf{true}]X \wedge [\mathit{iamaccessing}(t)]\Big( \\ \qquad \nu Y.\Big([\overline{\mathit{accessover}(t)}]Y \wedge \\ \qquad\quad \mu Z.(\langle\overline{\mathit{accessover}(t)} \wedge \overline{\mathit{iamaccessing}(t)}\rangle Z \vee \langle \mathit{accessover}(t)\rangle \mathsf{true}) \\ \qquad \Big) \\ \quad \Big)\end{array}$$

- Every thread eventually finishes its flush of a region:

$$\begin{array}{l}\forall t : \mathit{ThreadId}. \\ \quad \nu X.[\mathsf{true}]X \wedge [\mathit{iamflushing}(t)]\Big( \\ \qquad \nu Y.\Big([\overline{\mathit{flushover}(t)}]Y \wedge \\ \qquad\quad \mu Z.(\langle\overline{\mathit{flushover}(t)} \wedge \overline{\mathit{iamflushing}(t)}\rangle Z \vee \langle \mathit{flushover}(t)\rangle \mathsf{true}) \\ \qquad \Big) \\ \quad \Big)\end{array}$$

### A.1.3 Hesselink's Register

- Values that are written to the register can be read from the register if no other value is written to the register in the meantime.

$$\begin{array}{l}\nu X.[\mathsf{true}]X \wedge \forall w : D.[\mathit{begin\_write}(w)]\nu Y. \\ \qquad \Big([\overline{\mathit{end\_write}}]Y \wedge [\mathit{end\_write}]\nu Z. \\ \qquad\quad \Big([\overline{\exists d : D.\mathit{begin\_write}(d)}]Z \wedge [\mathit{begin\_read}]\nu W. \\ \qquad\qquad ([\overline{\exists d : D.\mathit{begin\_write}(d)}]W \wedge \\ \qquad\qquad\quad \forall w' : D.[\mathit{end\_read}(w')](w = w')) \\ \qquad\quad \Big) \\ \qquad \Big)\end{array}$$

### A.1.4 Elevator

For the elevator we verify the $\mu$-calculus version of the property that is checked in [FL09]. We verify whether, if a request is made for the top floor, denoted *storeys*, then the elevator

eventually is at the top floor.

$$\begin{array}{l}\nu X.\mu Y.\nu Z.[\mathit{move} \vee \mathit{close}]X \wedge \Big( \\ \quad \langle \mathit{isAt}(\mathit{storeys})\rangle \mathsf{true} \vee \Big( \\ \quad\quad [\mathit{move} \vee \mathit{close}]Z \wedge \\ \quad\quad (\neg(\langle \mathit{isPressed}(\mathit{storeys})\rangle \mathsf{true}) \vee [\mathit{move} \vee \mathit{close}]Y) \\ \quad \Big) \\ \Big)\end{array}$$

### A.1.5 Hanoi Towers

For the Hanoi towers we check whether a final configuration can be reached in a finite number of steps.

$$\mu X.\langle \overline{\mathit{done}} \rangle X \vee \langle \mathit{done} \rangle \mathsf{true}$$

### A.1.6 Two Player Board Games

For the two-player board games, we express for each player whether he has a winning strategy. For all games, these are variations of the following two formulae. Note that these formulae assume that the model of the game guarantees strict alternation between the two players.

- White has as winning strategy, assuming that the opponent makes the first move:

$$\mu X.\langle \mathit{WhiteWins} \rangle \mathsf{true} \vee [\mathsf{true}]\langle \mathsf{true} \rangle X$$

- Black has a winning strategy, assuming that he makes the first move:

$$\mu X.\langle \mathit{BlackWins} \rangle \mathsf{true} \vee \langle \mathsf{true} \rangle [\mathsf{true}] X$$

### A.1.7 IEEE 1394 Link-Layer

For the IEEE 1394 link-layer protocol we verify the properties described by Luttik [Lut97]. We here give a $\mu$-calculus version of the properties as they were described in ACTL in [SM98].

- P1394(n) is deadlock free:

$$\begin{array}{l}\nu X.[\mathsf{true}]X \wedge [\overline{\mathit{arbresgap} \vee (\exists l : \mathit{LDC}, \mathit{id} : \mathit{Nat}.\mathit{id} \leq N \wedge \mathit{LDcon}(\mathit{id}, l))}] \\ \quad (\nu Y.\langle \mathsf{true} \rangle \mathsf{true} \wedge [\exists l : \mathit{LDC}, \mathit{id} : \mathit{Nat}.\mathit{id} \leq N \wedge \mathit{LDcon}(\mathit{id}, l)]Y)\end{array}$$

  This denotes that the protocol is deadlock free under some added assumptions.

- Between two subsequent "subaction gap" signals at most two asynchronous packets have travelled over the BUS.

$$
\begin{aligned}
&\nu W.[\mathsf{true}]W \wedge [somecPDind]\nu X.\\
&\quad [\overline{somecPDind \vee \exists l : LDC.LDcon(0,l) \vee LDcon(1,l) \vee LDcon(2,l)}]X\\
&\wedge [\exists l : LDC.LDcon(0,l) \vee LDcon(1,l) \vee LDcon(2,l)]\nu Y.\\
&\quad\quad [\overline{somecPDind \vee \exists l : LDC.LDcon(0,l) \vee LDcon(1,l) \vee LDcon(2,l)}]Y\\
&\wedge [\exists l : LDC.LDcon(0,l) \vee LDcon(1,l) \vee LDcon(2,l)]\\
&\quad\quad \nu Z[\overline{somecPDind}]Z\\
&\wedge [\exists l : LDC.LDcon(0,l) \vee LDcon(1,l) \vee LDcon(2,l)]\mathsf{false}
\end{aligned}
$$

  Note that the occurrence of any "subaction gap" signal is encoded by the action *somecPDind*.

- If a node $0 \leq id \leq n-1$ emitted a request on the *LDreq* gate, and node $id$ communicates a request on the *PAreq* gate each time it receives a *subactgap* signal on the *PDind* gate (and before an *arbresgap* occurs), it also eventually receives a confirmation on the *LDcon* gate.

$$
\begin{aligned}
&\forall id : Nat.id \leq N \implies\\
&\quad \nu X.[\mathsf{true}]X \wedge\\
&\quad [(\exists n : Nat, h : HEADER, d : DATA.n \leq N \wedge LDreq(id,n,h,d))]\\
&\quad\quad \nu Y.[\overline{cPDind(id, subactgap) \vee arbresgap}]Y \wedge\\
&\quad\quad [(\exists p : PAR.cPAreq(id,p))]\\
&\quad\quad\quad \mu X.((\overline{\langle arbresgap\rangle}\mathsf{true} \wedge \overline{[arbresgap]X}\\
&\quad\quad\quad\quad \wedge \overline{[arbresgap \vee \exists l : LDC.LDcon(id,l)]\mathsf{false}}) \vee\\
&\quad\quad\quad (\overline{\langle \exists l : LDC.LDcon(id,l)\rangle}\mathsf{true} \wedge \overline{[\exists l : LDC.LDcon(id,l)]\mathsf{false}}))
\end{aligned}
$$

- Every request emitted by a node $0 \leq id \leq n-1$ on gate *PAreq* with parameter *immediate* is followed by a matching confirmation on gate *PAcon* with parameter *won*.

$$
\begin{aligned}
&\forall id : Nat.id \leq N \implies \nu X.[\mathsf{true}]X \wedge [cPAreq(id, immediate)]\\
&\quad \mu Y.\Big(\big(\langle cPAreq(id, immediate)\rangle\mathsf{true}\\
&\quad\quad \wedge [cPAreq(id, immediate)]Y\\
&\quad\quad \wedge \overline{[cPAreq(id, immediate) \vee cPAcon(id, won)]\mathsf{false}}\big)\\
&\quad \vee (\langle cPAcon(id, won)\rangle\mathsf{true} \wedge \overline{[cPAcon(id, won)]\mathsf{false}})\\
&\quad \Big)
\end{aligned}
$$

- Between two subsequent "arbitration reset gap" signals no node $0 \leq id \leq n-1$ receives a confirmation on gate *PAcon* with parameter *won* upon a request on gate *PAreq* with parameter *fair* more than once:

$$
\begin{aligned}
&\forall id : Nat.id \leq N \implies \nu X.[true]X \wedge [arbresgap]\\
&\quad \nu Y.[\overline{arbresgap}]Y \wedge [cPAreq(id, fair).cPAcon(id, won)]\\
&\quad\quad \nu Z.[\overline{arbresgap}]Z \wedge [cPAreq(id, fair).cPAcon(id, won)]\mathsf{false}
\end{aligned}
$$

### A.1.8 Truck Lift

For the truck lift, we verify the properties formulated in [GPW03].

- It is always possible for the system to get to a state in which pressing the *UP* button of any lift will yield the appropriate response:

$$\nu X.[\mathsf{true}]X \wedge \mu Y.(\langle \mathsf{true}\rangle Y \vee \langle \exists n : Nat.n < N \wedge up(n)\rangle \mathsf{true}$$

- It is always possible for the system to get to a state in which pressing the *DOWN* button of any lift will yield the appropriate response:

$$\nu X.[\mathsf{true}]X \wedge \mu Y.(\langle \mathsf{true}\rangle Y \vee \langle \exists n : Nat.n < N \wedge down(n)\rangle \mathsf{true}$$

- In any execution sequence contain only one button-pressed action, and containing no button-released action of the pressed button, the whole system always begins to move. This is the formulae for the *up* button.

$$\begin{array}{l}\nu W.[\overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id))}]W \wedge [up(address)] \\ \quad \nu X.([\overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id)) \vee released(address)}]X \wedge \\ \quad\quad \mu Y.(\langle \overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id) \vee released(address)}\rangle Y \vee \\ \quad\quad\quad \langle \exists id : Nat.id < N \wedge move(id, UP)\rangle \mathsf{true})))\end{array}$$

- In any execution sequence contain only one button-pressed action, and containing no button-released action of the pressed button, the whole system always begins to move. This is the formulae for the *down* button.

$$\begin{array}{l}\nu W.[\overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id))}]W \wedge [down(address)] \\ \quad \nu X.([\overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id)) \vee released(address)}]X \wedge \\ \quad\quad \mu Y.(\langle \overline{\exists id : Nat.id < N \wedge (up(id) \vee down(id) \vee released(address)}\rangle Y \vee \\ \quad\quad\quad \langle \exists id : Nat.id < N \wedge move(id, DOWN)\rangle \mathsf{true})))\end{array}$$

- As long as all other lifts move up, the last one cannot move down:

$$\begin{array}{l}\nu X(n : Nat = 0). \\ \quad ((n < N - 1 \implies ) \\ \quad\quad \nu X.[\overline{\exists id : Nat.id < N \wedge move(id, UP)}]X \wedge [move(n, UP)]X(n+1)) \wedge \\ \quad ((n = N - 1) \implies \\ \quad\quad \nu Y.[\overline{\exists id : Nat.id < N \wedge move(id, UP)}]Y \wedge [move(n, DOWN)]\mathsf{false})\end{array}$$

- If no *up* button is pressed, the system cannot move up:

$$\begin{array}{l}\nu X.[\overline{\exists id : Nat.id < N \wedge up(id)}]X \wedge \\ \quad [\exists id' : Nat.id' < N \wedge move(id', UP)]\mathsf{false}\end{array}$$

- If no *down* button is pressed, the system cannot move down:

$$\begin{array}{l}\nu X.[\overline{\exists id : Nat.id < N \wedge down(id)}]X \wedge \\ \quad [\exists id' : Nat.id' < N \wedge move(id', DOWN)]\mathsf{false}\end{array}$$

### A.1.9 Leader Election Protocol

For the leader election protocol we specify that it eventually stabilises.

$$
\begin{array}{l}
\nu X(n\colon \mathit{Int} = 0).\\
\quad ([\mathit{Add}]X(n+1) \wedge [\mathit{Remove}]X(n-1) \wedge [\tau \vee \mathit{Trigger}]X(n)\\
\quad \wedge (n = N \vee \mu Y.(\\
\qquad (\mu \mathit{Count}(i : \mathit{Nat} = 0).\\
\qquad\quad (i < n \wedge ([\mathit{leader}]\mathit{Count}(i+1)) \wedge (\langle \mathit{leader}\rangle \mathsf{true})) \vee\\
\qquad\quad (i = n \wedge [\mathit{leader}]\mathsf{false}))\\
\qquad \vee (\langle\tau\rangle \mathsf{true} \wedge [\tau]Y))))
\end{array}
$$

## A.2 Decision Procedures

In [FL09] classes of formulae are considered for validity and satisfiability checking. Furthermore, some formulae are provided as test cases for the tool MLsolver [FL09]. In our benchmarks, we consider the following classes of formulae that are checked for validity.

- The first class expresses that non-deterministic Büchi conditions and deterministic parity conditions are equi-expressive.

  $$\psi_n \triangleq \nu X. \bigcirc X \wedge \bigvee_{i=1}^{n} q_i \wedge \bigwedge_{j \neq i} \neg q_j$$

  is the subformula that expresses that in every state one of the propositions $q_i$ is true. The formula

  $$\begin{array}{l}\varphi_n \triangleq \psi_n \implies \big((\sigma X_n \ldots \nu X_2.\mu X_1. \bigwedge_{i=1}^{n} q_i \implies \bigcirc X_i) \iff\\ \quad \bigvee_{i \text{ even}} (\nu X.(\mu Y. q_i \vee \bigcirc Y) \wedge \bigcirc X) \wedge \bigwedge_{j>i, j \text{ odd}} \mu X.(\nu Y. \neg q_j \wedge \bigcirc Y) \vee \bigcirc X\big)\end{array}$$

  then expresses equi-expressivity of the conditions.

- We also consider the families of formulae where Kleene stars are nested in PDL. Let $\alpha_0 \triangleq \mathsf{true}?^*$ and $\alpha_{n+1} \triangleq (a^* \alpha_n b^*)^*$. This is used in the following three families of formulae:
  - $\varphi_n \triangleq \langle (a \cup b)^* \rangle q \vee [\alpha_n] \neg q$.
  - $\psi_n \triangleq \langle \alpha_n \rangle q \vee [(a \cup b)^*] \neg q$.
  - $\gamma_n \triangleq \langle \alpha_n \rangle q \vee [\alpha_n] \neg q$.
- The limit of an infinite sequence of prefix-sharing paths in a system is, again, a path in the system. This is referred to as the limit closure. For $\mathsf{CTL}^*$, this can be expressed as $LC^*(\varphi, \psi) \triangleq AG(E\psi \implies EX((E\varphi)UE\psi)) \wedge E\psi \implies EG((E\varphi)UE\psi)$. We consider, among others, the formula $LC^*(\varphi_n, \psi_n)$, where $\varphi_n \triangleq G(\bigvee_{i \leq n} \neg q_i)$, $\psi_0 \triangleq q_0$, $\psi_{2n+1} \triangleq q_{2n+1} \wedge X\psi_{2n}$, and $\psi_{2n+2} \triangleq q_{2n+2} \vee X\psi_{2n+1}$. We consider similar formulae for $\mathsf{CTL}$.

In addition some simpler formulae are considered, as well as formulae that describe binary counters and a fair scheduler. For the details of those we refer to the implementation of the benchmarks.

# Bibliography


[ACM07] ACM. *Turing Award Honors Founders of Automatic Verification Technology*. 2007. URL: http://www.acm.org/press-room/awards/turing-award-07 (visited on 28/02/2013).

[Adl07] I. Adler. 'Directed tree-width examples'. In: *Journal of Combinatorial Theory, Series B* 97.5 (2007), pp. 718–725. DOI: 10.1016/j.jctb.2006.12.006.

[ASU86] A.V. Aho, R. Sethi and J.D. Ullman. *Compilers: Principles, Techniques, and Tools (first edition)*. Addison-Wesley, 1986, p. 796. ISBN: 0-201-10088-6.

[Alb+05] M.H. Albert et al. 'An introduction to clobber.' In: *Integers* 5.2 (2005). URL: http://eudml.org/doc/129044.

[AHK02] R. Alur, T.A. Henzinger and O. Kupferman. 'Alternating-time temporal logic'. In: *Journal of the ACM* 49.5 (2002), pp. 672–713. DOI: 10.1145/585265.585270.

[And93] H.R. Andersen. 'Verification of Temporal Properties of Concurrent Systems'. PhD thesis. Aarhus University, 1993, p. 283. URL: http://ojs.statsbiblioteket.dk/index.php/daimipb/article/view/6762.

[And94] H.R. Andersen. 'Model checking and Boolean graphs'. In: *Theoretical Computer Science* 126.1 (1994), pp. 3–30. DOI: 10.1016/0304-3975(94)90266-6.

[ACP87] S. Arnborg, D.G. Corneil and A. Proskurowski. 'Complexity of Finding Embeddings in a k-Tree'. In: *SIAM Journal on Algebraic Discrete Methods* 8.2 (1987), pp. 277–284. DOI: 10.1137/0608024.

[AC88] A. Arnold and P. Crubille. 'A Linear Algorithm to Solve Fixed-Point Equations on Transition Systems'. In: *Information Processing Letters* 20.1 (1988), pp. 57–66. DOI: 10.1016/0020-0190(88)90029-4.

[AVW03] A. Arnold, A. Vincent and I. Walukiewicz. 'Games for synthesis of controllers with partial observation'. In: *Theoretical Computer Science* 303.1 (2003), pp. 7–34. DOI: 10.1016/S0304-3975(02)00442-5.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008, p. 984. ISBN: 9780262026499.

[Bar06] J. Barát. 'Directed Path-width and Monotonicity in Digraph Searching'. In: *Graphs and Combinatorics* 22.2 (2006), pp. 161–172. DOI: 10.1007/s00373-005-0627-y.

[BSW69] K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson. 'A note on reliable full-duplex transmission over half-duplex links'. In: *Communications of the ACM* 12.5 (1969), pp. 260–261. DOI: 10.1145/362946.362970.

[Bas96] T. Basten. 'Branching bisimilarity is an equivalence indeed!' In: *Information Processing Letters* 58.3 (1996), pp. 141–147. DOI: 10.1016/0020-0190(96)00034-8.

[BBC00] A. Beck, M.N. Bleicher and D.W. Crowe. *Excursions into Mathematics: The Millennium Edition*. CRC Press, 2000. ISBN: 1568811152.

[BV01] E. Beffara and S.G. Vorobyov. *Adapting Gurvich-Karzanov-Khachiyan's Algorithm for Parity Games*. Tech. rep. Uppsala: Uppsala University, Sweden, 2001. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-40715.

[BK84] J.A. Bergstra and J.W. Klop. 'Process algebra for synchronous communication'. In: *Information and Control* 60.1-3 (1984), pp. 109–137. DOI: 10.1016/S0019-9958(84)80025-X.

[BC04] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004. ISBN: 978-3-540-20854-9.

[BG04] D. Berwanger and E. Grädel. 'Fixed-Point Logics and Solitaire Games'. In: *Theory of Computing Systems* 37.6 (2004), pp. 675–694. DOI: 10.1007/s00224-004-1147-5.

[BG05] D. Berwanger and E. Grädel. 'Entanglement — A Measure for the Complexity of Directed Graphs with Applications to Logic and Games'. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Vol. 3452. LNCS. Springer, 2005, pp. 209–223. DOI: 10.1007/978-3-540-32275-7_15.

[Ber+06] D. Berwanger et al. 'DAG-width and parity games'. In: *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*. Vol. 3884. LNCS. Springer, 2006, pp. 436–524. DOI: 10.1007/11672142_43.

[Ber+12a] D. Berwanger et al. 'Entanglement and the complexity of directed graphs'. In: *Theoretical Computer Science* 463 (2012), pp. 2–25. DOI: 10.1016/j.tcs.2012.07.010.

[Ber+12b] D. Berwanger et al. 'The DAG-width of directed graphs'. In: *Journal of Combinatorial Theory, Series B* 102.4 (2012), pp. 900–923. DOI: 10.1016/j.jctb.2012.04.004.

[Bie+99] A. Biere et al. 'Symbolic Model Checking without BDDs'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*. Vol. 1579. LNCS. Springer, 1999, pp. 193–207. DOI: 10.1007/3-540-49059-0_14.

[BSV03] H. Björklund, S. Sandberg and S.G. Vorobyov. 'A Discrete Subexponential Algorithm for Parity Games'. In: *STACS 2003*. Vol. 2607. LNCS. Springer, 2003, pp. 663–674. DOI: 10.1007/3-540-36494-3_58.

[BV07] H. Björklund and S.G. Vorobyov. 'A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games'. In: *Discrete Applied Mathematics* 155.2 (2007), pp. 210–229. DOI: 10.1016/j.dam.2006.04.029.

[BO03] S.C.C. Blom and S. Orzan. 'Distributed Branching Bisimulation Reduction of State Spaces'. In: *PDMC 2003*. Vol. 89. 1. Elsevier, 2003, pp. 99–113. DOI: 10.1016/S1571-0661(05)80099-4.

[BP02] S.C.C. Blom and J.C. van de Pol. 'State Space Reduction by Proving Confluence'. In: *Computer Aided Verification*. Vol. 2404. LNCS. Springer, 2002, pp. 596–609. DOI: 10.1007/3-540-45657-0_50.

[Bod97] H.L. Bodlaender. 'Treewidth: Algorithmic techniques and results'. In: *Mathematical Foundations of Computer Science 1997*. Vol. 1295. LNCS. Springer, 1997, pp. 19–36. DOI: 10.1007/BFb0029946.

[BK10] H.L. Bodlaender and A.M.C.A. Koster. 'Treewidth computations I. Upper bounds'. In: *Information and Computation* 208.3 (2010), pp. 259–275. DOI: 10.1016/j.ic.2009.03.008.

[BK11] H.L. Bodlaender and A.M.C.A. Koster. 'Treewidth computations II. Lower bounds'. In: *Information and Computation* 209.7 (2011), pp. 1103–1119. DOI: 10.1016/j.ic.2011.04.003.

[Bra91] J.C. Bradfield. 'Verifying Temporal Properties of Systems with Applications to Petri Nets'. PhD thesis. University of Edinburgh, 1991. URL: http://hdl.handle.net/1842/6565.

[BS00] J.C. Bradfield and C. Stirling. 'Modal Logics and mu-Calculi: an Introduction'. In: *Handbook of Process Algebra*. Elsevier, 2000. Chap. 4, pp. 293–330. DOI: 10.1016/B978-044482830-9/50022-9.

[Bre01] E.A. Bretz. 'By-wire cars turn the corner'. In: *IEEE Spectrum* 38.4 (2001), pp. 68–73. DOI: 10.1109/6.915192.

[BHR84] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. 'A Theory of Communicating Sequential Processes'. In: *J. ACM* 31.3 (June 1984), pp. 560–599. DOI: 10.1145/828.833.

[BCG88] M.C. Browne, E.M. Clarke and O. Grumberg. 'Characterizing Finite Kripke Structures in Propositional Temporal Logic'. In: *Theoretical Computer Science* 59 (1988), pp. 115–131. DOI: 10.1016/0304-3975(88)90098-9.

[Bry86] R.E. Bryant. 'Graph-Based Algorithms for Boolean Function Manipulation'. In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[Bur+90] J.R. Burch et al. 'Symbolic model checking: $10^{20}$ states and beyond'. In: *LICS'90 Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*. IEEE Comput. Soc. Press, 1990, pp. 428–439. DOI: 10.1109/LICS.1990.113767.

[CGR12] F. Canavoi, E. Grädel and R. Rabinovich. 'The discrete strategy improvement algorithm for parity games and complexity measures for directed graphs'. In: *Proceedings GandALF 2012*. Vol. 96. Electronic Proceedings in Theoretical Computer Science 1. 2012, pp. 197–209. DOI: 10.4204/EPTCS.96.15.

[CK74] V. Cerf and R.E. Kahn. 'A Protocol for Packet Network Intercommunication'. In: *IEEE Transactions on Communications* 22.5 (1974), pp. 637–648. DOI: 10.1109/TCOM.1974.1092259.

[Cha09] R.N. Charette. 'This Car Runs on Code'. In: *IEEE Spectrum online* (Feb. 2009). URL: http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code.

[Cha+10] K. Chatterjee et al. 'GIST: A Solver for Probabilistic Games'. In: *CAV 2012*. Vol. 6174. LNCS. Springer, 2010, pp. 665–669. DOI: 10.1007/978-3-642-14295-6_57.

[Che+07] T. Chen et al. 'Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems'. In: *CONCUR'07: Concurrency Theory*. Vol. 4703. LNCS. Springer, 2007, pp. 120–135. DOI: 10.1007/978-3-540-74407-8_9.

[Cim+00] A. Cimatti et al. 'NUSMV: a new symbolic model checker'. In: *International Journal on Software Tools for Technology Transfer (STTT)* 2.4 (2000), pp. 410–425. DOI: 10.1007/s100090050046.

[CE82] E.M. Clarke and E.A. Emerson. 'Design and synthesis of synchronization skeletons using branching time temporal logic'. In: *Logics of Programs*. Vol. 131. LNCS. Springer, 1982, pp. 52–71. DOI: 10.1007/BFb0025774.

[CES86] E.M. Clarke, E.A. Emerson and A.P. Sistla. 'Automatic verification of finite-state concurrent systems using temporal logic specifications'. In: *ACM Transactions on Programming Languages and Systems* 8.2 (1986), pp. 244–263. DOI: 10.1145/5397.5399.

[CKS93] R. Cleaveland, M. Klein and B. Steffen. 'Faster model checking for the modal mu-Calculus'. In: *Computer Aided Verification*. Vol. 663. LNCS. Springer, 1993, pp. 410–422. DOI: 10.1007/3-540-56496-9_32.

[CPS93] R. Cleaveland, J. Parrow and B. Steffen. 'The concurrency workbench: a semantics-based tool for the verification of concurrent systems'. In: *ACM Transactions on Programming Languages and Systems* 15.1 (1993), pp. 36–72. DOI: 10.1145/151646.151648.

[CS01] R. Cleaveland and O. Sokolsky. 'Equivalence and Preorder Checking for Finite-State Systems'. In: *Handbook of Process Algebra*. Elsevier, 2001, pp. 391–424. DOI: 10.1016/B978-044482830-9/50024-2.

[CS91] R. Cleaveland and B. Steffen. 'Computing behavioural relations, logically'. In: *Automata, Languages and Programming*. Vol. 510. LNCS. Springer, 1991, pp. 127–138. DOI: 10.1007/3-540-54233-7_129.

[Col99] R.P.G. Collinson. 'Fly-by-wire flight control'. In: *Computing Control Engineering Journal* 10.4 (1999), pp. 141–152. DOI: 10.1049/cce:19990403.

[CO00] B. Courcelle and S. Olariu. 'Upper bounds to the clique width of graphs'. In: *Discrete Applied Mathematics* 101.1-3 (2000), pp. 77–114. DOI: 10.1016/S0166-218X(99)00184-5.

[CKW11] S. Cranen, J.J.A. Keiren and T.A.C. Willemse. 'Stuttering Mostly Speeds Up Solving Parity Games'. In: *NASA Formal Methods*. Vol. 6617. LNCS. Springer, 2011, pp. 207–221. DOI: 10.1007/978-3-642-20398-5_16. eprint: 1102.2366.

[CKW12a] S. Cranen, J.J.A. Keiren and T.A.C. Willemse. *A cure for stuttering parity games*. Tech. rep. 12-05. Eindhoven: Eindhoven University of Technology, 2012, p. 25. URL: http://alexandria.tue.nl/repository/books/732149.pdf.

[CKW12b] S. Cranen, J.J.A. Keiren and T.A.C. Willemse. 'A cure for stuttering parity games'. In: *Theoretical Aspects of Computing — ICTAC 2012*. Vol. 7521. LNCS. Springer, 2012, pp. 198–212. DOI: 10.1007/978-3-642-32943-2_16.

[Cra+13] S. Cranen et al. 'An Overview of the mCRL2 Toolset and its Recent Advances'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*. Vol. 7795. LNCS. Springer, 2013, pp. 199–213. DOI: 10.1007/978-3-642-36742-7_15.

[DPW08] A. van Dam, S.C.W. Ploeger and T.A.C. Willemse. 'Instantiation for Parameterised Boolean Equation Systems'. In: *Theoretical Aspects of Computing — ICTAC 2008*. Vol. 5160. LNCS 642. Springer, 2008, pp. 440–454. DOI: 10.1007/978-3-540-85762-4_30.

[DKT97] N.D. Dendris, L.M. Kirousis and D.M. Thilikos. 'Fugitive-search games on graphs and related parameters'. In: *Theoretical Computer Science* 172.1-2 (1997), pp. 233–254. DOI: 10.1016/S0304-3975(96)00177-6.

[Dic86] A. Dicky. 'An algebraic and algorithmic method for analysing transition systems'. In: *Theoretical Computer Science* 46 (1986), pp. 285–303. DOI: 10.1016/0304-3975(86)90034-4.

[Dij75] E.W. Dijkstra. 'Guarded commands, nondeterminacy and formal derivation of programs'. In: *Communications of the ACM* 18.8 (1975), pp. 453–457. DOI: 10.1145/360933.360975.

[Dij76] E.W. Dijkstra. 'A Discipline of Programming'. In: Prentice-Hall, 1976. ISBN: 013215871X.

[DM06] B. Dutertre and L. de Moura. *The YICES SMT Solver*. 2006. URL: http://yices.csl.sri.com/tool-paper.pdf (visited on 18/07/2013).

[EJ09] C. Ebert and C. Jones. 'Embedded Software: Facts, Figures, and Future'. In: *Computer* 42.4 (2009), pp. 42–52. DOI: 10.1109/MC.2009.118.

[ES04] N. Eén and N. Sörensson. 'An Extensible SAT-solver'. In: *Theory and Applications of Satisfiability Testing*. Vol. 2919. LNCS. Springer, 2004, pp. 333–336. DOI: 10.1007/978-3-540-24605-3_37.

[EC80] E.A. Emerson and E.M. Clarke. 'Characterizing correctness properties of parallel programs using fixpoints'. In: *Automata, Languages and Programming*. Vol. 85. LNCS. Springer, 1980, pp. 169–181. DOI: 10.1007/3-540-10003-2_69.

[EH86] E.A. Emerson and J.Y. Halpern. '"Sometimes" and "not never" revisited: on branching versus linear time temporal logic'. In: *Journal of the ACM* 33.1 (1986), pp. 151–178. DOI: 10.1145/4904.4999.

[EJ91] E.A. Emerson and C.S. Jutla. 'Tree automata, Mu-Calculus and determinacy'. In: *SFCS '91: Proceedings of the 32nd annual symposium on Foundations of computer science*. IEEE Computer Society, 1991, pp. 368–377. DOI: 10.1109/SFCS.1991.185392.

[EJ99] E.A. Emerson and C.S. Jutla. 'The Complexity of Tree Automata and Logics of Programs'. In: *SIAM Journal on Computing* 29.1 (1999), pp. 132–158. DOI: 10.1137/S0097539793304741.

[EL86] E.A. Emerson and C.L.L. Lei. 'Efficient Model Checking in Fragments of the Propositional Mu-Calculus'. In: *Proceedings of LICS 1986*. IEEE Computer Society, 1986, pp. 267–278. ISBN: 0-8186-0720-3.

[FBG03] J.C. Fernandez, M. Bozga and L. Ghirvu. 'State space reduction based on live variables analysis'. In: *Science of Computer Programming* 47.2-3 (2003), pp. 203–220. DOI: 10.1016/S0167-6423(02)00133-8.

[Flo67] R.W. Floyd. 'Assigning meanings to programs'. In: *Proceedings of a Symposium on Applied Mathematics*. Vol. 19. Mathematical Aspects of Computer Science. American Mathematical Society, 1967, pp. 19–32. URL: http://www.eecs.berkeley.edu/~necula/Papers/FloydMeaning.pdf.

[FP03] W.J. Fokkink and J. Pang. 'Cones and Foci for Protocol Verification Revisited'. In: *Foundations of Software Science and Computation Structures*. Vol. 2620. LNCS. Springer, 2003, pp. 267–281. DOI: 10.1007/3-540-36576-1_17.

[FPP06] W.J. Fokkink, J. Pang and J.C. van de Pol. 'Cones and foci: A mechanical framework for protocol verification'. In: *Formal Methods in System Design* 29.1 (2006), pp. 1–31. DOI: 10.1007/s10703-006-0004-3.

[For] Formal Systems Europe Limited. *FDR2 Manual*. URL: http://www.fsel.com/fdr2_manual.html (visited on 12/03/2013).

[Fri09] O. Friedmann. 'A Super-Polynomial Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it'. In: *2009 24th Annual IEEE Symposium on Logic In Computer Science* 7 (2009), pp. 145–156. DOI: 10.1109/LICS.2009.27. arXiv:1106.0778.

[Fri10] O. Friedmann. 'The Stevens-Stirling-Algorithm for Solving Parity Games Locally Requires Exponential Time'. In: *International Journal of Foundations of Computer Science* 21.03 (2010), pp. 277–287. DOI: 10.1142/S0129054110007246.

[Fri11a] O. Friedmann. 'An Exponential Lower Bound for the Latest Deterministic Strategy Iteration Algorithms'. In: *Logical Methods in Computer Science* 7 (2011), pp. 1–42. DOI: 10.2168/LMCS-7(3:23)2011. arXiv:1106.0778.

[Fri11b] O. Friedmann. 'Exponential Lower Bounds for Solving Infinitary Payoff Games and Linear Programs'. PhD thesis. Ludwig-Maximilians-Universität München, 2011. URL: http://nbn-resolving.de/urn:nbn:de:bvb:19-132940.

[Fri11c] O. Friedmann. 'Recursive algorithm for parity games requires exponential time'. In: *RAIRO - Theoretical Informatics and Applications* 45.4 (2011), pp. 449–457. DOI: 10.1051/ita/2011124.

[Fri13] O. Friedmann. 'A Subexponential Lower Bound for Policy Iteration based on Snare Memorization'. In: *Discrete Applied Mathematics* 161 (10–11 2013). DOI: 10.1016/j.dam.2013.02.007.

[FHZ11] O. Friedmann, T.D. Hansen and U. Zwick. 'A subexponential lower bound for the Random Facet algorithm for Parity Games'. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*. SIAM, 2011, pp. 202–216. URL: http://www.siam.org/proceedings/soda/2011/SODA11_019_friedmanno.pdf.

[FL09] O. Friedmann and M. Lange. 'Solving Parity Games in Practice'. In: *Automated Technology for Verification and Analysis 7th International Symposium, ATVA 2009*. Vol. 5799. LNCS. Springer, 2009, pp. 182–196. DOI: 10.1007/978-3-642-04761-9_15.

[FL10a] O. Friedmann and M. Lange. 'A Solver for Modal Fixpoint Logics'. In: *Electronic Notes in Theoretical Computer Science*. Vol. 262. Elsevier, 2010, pp. 99–111. DOI: 10.1016/j.entcs.2010.04.008.

[FL10b] O. Friedmann and M. Lange. *The PGSolver Collection of Parity Game Solvers*. Tech. rep. Institut für Informatik, Ludwig-Maximilians-Universität München, Germany, 2010, pp. 1–61. URL: https://github.com/oliverfriedmann/pgsolver/blob/master/doc/pgsolver.pdf (visited on 18/07/2013).

[FL12] O. Friedmann and M. Lange. 'Two local strategy iteration schemes for parity game solving'. In: *International Journal of Foundations of Computer Science* 23.3 (2012), pp. 669–685. DOI: 10.1142/S0129054112400333.

[FLL10] O. Friedmann, M. Latte and M. Lange. 'A Decision Procedure for CTL$^*$ Based on Tableaux and Automata'. In: *Automated Reasoning*. Vol. 6173. LNCS. Springer, 2010, pp. 331–345. DOI: 10.1007/978-3-642-14203-1_28.

[FW06] C. Fritz and T. Wilke. 'Simulation Relations for Alternating Parity Automata and Parity Games'. In: *Developments in Language Theory*. Vol. 4036. LNCS. Springer, 2006, pp. 59–70. DOI: 10.1007/11779148_7.

[Gar+11] H. Garavel et al. 'CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes'. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 6605. LNCS. Springer, 2011, pp. 372–387. DOI: 10.1007/978-3-642-19835-9_33.

[Gar74] M. Gardner. 'Mathematical Games: Cram, Crosscram and Quadraphage: New Games having Elusive Winning Strategies.' In: *Scientific American* 230 (1974), pp. 106–108.

[GS08] T. Gawlitza and H. Seidl. 'Precise Interval Analysis vs. Parity Games'. In: *FM '08: Proceedings of the 15th international symposium on Formal Methods*. Vol. 5014. LNCS. Springer, 2008, pp. 342–357. DOI: 10.1007/978-3-540-68237-0_24.

[GW12] M.W. Gazda and T.A.C. Willemse. 'Consistent Consequence for Boolean Equation Systems'. In: *SOFSEM 2012: Theory and Practice of Computer Science*. Vol. 7147. LNCS. Springer, 2012, pp. 277–288. DOI: 10.1007/978-3-642-27660-6_23.

[GW13] M.W. Gazda and T.A.C. Willemse. In: *Proceedings GandALF 2013*. Vol. 119. EPTCS. 2013, pp. 7–20. DOI: 10.4204/EPTCS.119.4.

[Gla90] R.J. van Glabbeek. 'The linear time - branching time spectrum'. In: *CONCUR '90 Theories of Concurrency: Unification and Extension*. Vol. 458. LNCS. Springer, 1990, pp. 278–297. DOI: 10.1007/BFb0039066.

[Gla93] R.J. van Glabbeek. 'The linear time - Branching time spectrum II'. In: *CONCUR'93: Concurrency theory*. Vol. 715. LNCS. Springer, 1993, pp. 66–81. DOI: 10.1007/3-540-57208-2_6.

[GW96] R.J. van Glabbeek and W.P. Weijland. 'Branching time and abstraction in bisimulation semantics'. In: *Journal of the ACM* 43.3 (1996), pp. 555–600. DOI: 10.1145/233551.233556.

[GD04] V. Gogate and R. Dechter. 'A complete anytime algorithm for treewidth'. In: *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. UAI '04. AUAI Press, 2004, pp. 201–208. ISBN: 0-9749039-0-6. URL: http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1109&proceeding_id=20.

[GN47] H.H. Goldstine and J. von Neumann. *Planning and coding problems for an electronic computer instrument*. Tech. rep. 1946-57, nos. 4,8,11. Insitute for Advanced Study Princeton, 1947. URL: http://archive.org/stream/planningcodingof0103inst#page/n5/mode/2up.

[GLS01] G. Gottlob, N. Leone and F. Scarcello. 'Robbers, marshals, and guards'. In: *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '01*. ACM Press, 2001, pp. 195–206. DOI: 10.1145/375551.375579.

[GTW02] E. Grädel, W. Thomas and T. Wilke, eds. *Automata, Logics and Infinite Games*. Springer, 2002. ISBN: 978-3-540-00388-5.

[GK05] J.F. Groote and M.K. Keinänen. 'A Sub-quadratic Algorithm for Conjunctive and Disjunctive Boolean Equation Systems'. In: *Theoretical Aspects of Computing – ICTAC 2005*. Vol. 3722. LNCS. Springer, 2005, pp. 532–545. DOI: 10.1007/11560647_35.

[GL02] J.F. Groote and B. Lisser. 'Computer assisted manipulation of algebraic process specifications'. In: *ACM SIGPLAN Notices* 37.12 (2002), p. 98. DOI: 10.1145/636517.636531.

[GPW03] J.F. Groote, J. Pang and A.G.G. Wouters. 'Analysis of a distributed system for lifting trucks'. In: *The Journal of Logic and Algebraic Programming* 55.1-2 (2003), pp. 21–56. DOI: 10.1016/S1567-8326(02)00038-3.

[GP09] J.F. Groote and S.C.W. Ploeger. 'Switching Graphs'. In: *International Journal of Foundations of Computer Science (IJFCS)* 20.5 (2009), pp. 869–886. DOI: 10.1142/S0129054109006930.

[GP96] J.F. Groote and J. van de Pol. 'A bounded retransmission protocol for large data packets'. In: *Algebraic Methodology and Software Technology*. Vol. 1101. LNCS. Springer, 1996, pp. 536–550. DOI: 10.1007/BFb0014338.

[GP00] J.F. Groote and J.C. van de Pol. 'State Space Reduction Using Partial $\tau$-Confluence'. In: *Mathematical Foundations of Computer Science 2000*. Vol. 1893. LNCS. Springer, 2000, pp. 383–393. DOI: 10.1007/3-540-44612-5_34.

[GS95] J.F. Groote and M.P.A. Sellink. 'Confluence for process verification'. In: *CONCUR '95: Concurrency Theory*. Vol. 962. LNCS. Springer, 1995, pp. 204–218. DOI: 10.1007/3-540-60218-6_15.

[GS96] J.F. Groote and M.P.A. Sellink. 'Confluence for process verification'. In: *Theoretical Computer Science* 170.1-2 (1996), pp. 47–81. DOI: 10.1016/S0304-3975(96)80702-X.

[GS01] J.F. Groote and J. Springintveld. 'Focus points and convergent process operators: a proof strategy for protocol verification'. In: *The Journal of Logic and Algebraic Programming* 49.1-2 (2001), pp. 31–60. DOI: 10.1016/S1567-8326(01)00010-8.

[GV90] J.F. Groote and F.W. Vaandrager. 'An efficient algorithm for branching bisimulation and stuttering equivalence'. In: *Automata, Languages and Programming*. Vol. 443. LNCS. Springer, 1990, pp. 626–638. DOI: 10.1007/BFb0032063.

[GW05a] J.F. Groote and T.A.C. Willemse. 'Model-checking processes with data'. In: *Science of Computer Programming* 56.3 (2005), pp. 251–273. DOI: 10.1016/j.scico.2004.08.002.

[GW05b] J.F. Groote and T.A.C. Willemse. 'Parameterised Boolean equation systems'. In: *Theoretical Computer Science* 343.3 (2005), pp. 332–369. DOI: 10.1016/j.tcs.2005.06.016.

[Gro+11] J.F. Groote et al. 'Experiences in developing the mCRL2 toolset'. In: *Software: Practice and Experience* 41.2 (2011), pp. 143–153. DOI: 10.1002/spe.1021.

[GH82] Y. Gurevich and L. Harrington. 'Trees, automata, and games'. In: *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*. ACM, 1982, pp. 60–65. DOI: 10.1145/800070.802177.

[Har09] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. 1st. Cambridge University Press, 2009. ISBN: 9780521899574.

[Hea+08] J. Heath et al. 'Probabilistic model checking of complex biological pathways'. In: *Theoretical Computer Science* 391.3 (2008), pp. 239–257. DOI: 10.1016/j.tcs.2007.11.013.

[HM97] N. Heintze and D. McAllester. 'Linear-time subtransitive control flow analysis'. In: *Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation - PLDI '97*. ACM Press, 1997, pp. 261–272. DOI: 10.1145/258915.258939.

[Hes98] W.H. Hesselink. 'Invariants for the construction of a handshake register'. In: *Information Processing Letters* 68 (1998), pp. 173–177. DOI: 10.1016/S0020-0190(98)00158-6.

[Hoa69] C.A.R. Hoare. 'An axiomatic basis for computer programming'. In: *Communications of the ACM* 12.10 (1969), pp. 576–580. DOI: 10.1145/363235.363259.

[Hoa80] C.A.R. Hoare. 'A Model for Communicating Sequential Processes'. In: *On the Construction of Programs*. 1980, pp. 229–254.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985. ISBN: 0131532715.

[Hun07] P.W. Hunter. 'Complexity and infinite games on finite graphs'. PhD thesis. University of Cambridge, 2007. URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-704.pdf.

[HK08] P.W. Hunter and S. Kreutzer. 'Digraph measures: Kelly decompositions, games, and orderings'. In: *Theoretical Computer Science* 399.3 (2008), pp. 206–219. DOI: 10.1016/j.tcs.2008.02.038.

[Hwo+13] Y.L. Hwong et al. 'Formalising and Analysing the Control Software of the Compact Muon Solenoid Experiment at the Large Hadron Collider'. In: *Science of Computer Programming* (2013). DOI: 10.1016/j.scico.2012.11.009.

[Joh+01] T. Johnson et al. 'Directed Tree-Width'. In: *Journal of Combinatorial Theory, Series B* 82.1 (2001), pp. 138–154. DOI: 10.1006/jctb.2000.2031.

[Jur98] M. Jurdziński. 'Deciding the winner in parity games is in UP ∩ co-UP'. In: *Information Processing Letters* 68.3 (1998), pp. 119–124. DOI: 10.1016/S0020-0190(98)00150-1.

[Jur00] M. Jurdziński. 'Small Progress Measures for Solving Parity Games'. In: *STACS '00: Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*. Vol. 1770. LNCS. Springer, 2000, pp. 290–301. DOI: 10.1007/3-540-46541-3_24.

[JPZ06] M. Jurdziński, M. Paterson and U. Zwick. 'A deterministic subexponential algorithm for solving parity games'. In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06* (2006), pp. 117–123. DOI: 10.1145/1109557.1109571.

[Kei06] M.K. Keinänen. 'Solving Boolean Equation Systems'. PhD thesis. Helsinki University of Technology, Department of Computer Science and Engineering, 2006. ISBN: 9512285460. URL: http://lib.tkk.fi/Diss/2006/isbn9512285460.

[Kei09] J.J.A. Keiren. 'An experimental study of algorithms and optimisations for parity games, with an application to Boolean Equation Systems'. MA thesis. Eindhoven University of Technology, 2009. URL: http://alexandria.tue.nl/extra1/afstversl/wsk-i/keiren2009.pdf.

[Kei13] J.J.A. Keiren. *Reduction and Solving of Parity Games*. Dataset. 2013. DOI: 10.4121/uuid:ff67b0cb-91e0-4c14-a622-73219b9a8fc2.

[KK12] J.J.A. Keiren and M.D. Klabbers. 'Modelling and verifying IEEE Std 11073-20601 session setup using mCRL2'. In: *Proc. AVoCS 2012*. Vol. X. Electronic Communications of the EASST. EASST, 2012, pp. 1–15. URL: http://journal.ub.tu-berlin.de/eceasst/article/view/793.

[KRW12] J.J.A. Keiren, M.A. Reniers and T.A.C. Willemse. 'Structural Analysis of Boolean Equation Systems'. In: *ACM Transactions on Computational Logic* 13.1 (2012), pp. 1–35. DOI: 10.1145/2071368.2071376.

[KWW13] J.J.A. Keiren, J.W. Wesselink and T.A.C. Willemse. *Improved Static Analysis of Parameterised Boolean Equation Systems using Control Flow Reconstruction*. Under submission. 2013. URL: http://arxiv.org/abs/1304.6482.

[KW11] J.J.A. Keiren and T.A.C. Willemse. 'Bisimulation Minimisations for Boolean Equation Systems'. In: *Proceedings of Haifa Verification Conference 2009 (HVC'09)*. Vol. 6405. LNCS. Springer, Heidelberg, 2011, pp. 102–116. DOI: 10.1007/978-3-642-19237-1_12.

[Kna28] B. Knaster. 'Un théorème sur les fonctions d'ensembles'. In: *Annales de la Société Polonaise de Mathématiques* 6 (1928), pp. 133–134.

[KM90] C.P.J. Koymans and J.C. Mulder. 'A modular approach to protocol verification using process algebra'. In: *Applications of Process Algebra*. Cambridge Tracts in Theoretical Computer Science 17. 1990, pp. 261–306. DOI: 10.1017/CBO9780511608841.012.

[Koz83] D. Kozen. 'Results on the propositional $\mu$-calculus'. In: *Theoretical Computer Science* 27.3 (1983), pp. 333–354. DOI: 10.1016/0304-3975(82)90125-6.

[KRS01] K.N. Kumar, C.R. Ramakrishnan and S.A. Smolka. 'Alternating Fixed Points in Boolean Equation Systems as Preferred Stable Models'. In: *Proceedings of the 17th International Conference on Logic Programming*. Springer, Nov. 2001, pp. 227–241. DOI: 10.1007/3-540-45635-X_23.

[KNP11] M. Kwiatkowska, G. Norman and D. Parker. 'PRISM 4.0: Verification of Probabilistic Real-time Systems'. In: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*. Vol. 6806. LNCS. Springer, 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47.

[Lan05] M. Lange. 'Solving parity games by a reduction to SAT". In: *In Proc. of the Workshop on Games in Design and Verification, GDV'05*. 2005.

[Lar93] K.G. Larsen. 'Efficient Local Correctness Checking'. In: *Computer Aided Verification, Fourth International Workshop, CAV '92, Proceedings*. Vol. 663. LNCS. Springer, 1993, pp. 30–43. DOI: 10.1007/3-540-56496-9_4.

[LNS82] J.-L. Lassez, V.L. Nguyen and E.A. Sonenberg. 'Fixed point theorems and semantics: a folk tale'. In: *Information Processing Letters* 14.3 (1982), pp. 112–116. DOI: 10.1016/0020-0190(82)90065-5.

[LRS98] X. Liu, C.R. Ramakrishnan and S.A. Smolka. 'Fully Local and Efficient Evaluation of Alternating Fixed Points'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*. Vol. 1384. LNCS. Springer, 1998, pp. 5–19. DOI: 10.1007/BFb0054161.

[LS98] X. Liu and S.A. Smolka. 'Simple Linear-Time Algorithms for Minimal Fixed Points'. In: *Proceedings of ICALP'98*. Vol. 1443. LNCS. Springer, 1998, pp. 53–66. DOI: 10.1007/BFb0055040.

[Lut97] S.P. Luttik. 'Description and Formal Specification of the Link Layer of P1394'. In: *Workshop on Applied Formal Methods in System Design*. 1997, pp. 43–56.

[Maa05] T. Maarup. 'Hex - Everything You Always Wanted to Know About Hex but Were Afraid to Ask'. MA thesis. Department of Mathematics and Computer Science, University of Southern Denmark, 2005. URL: http://maarup.net/thomas/hex/hex3.pdf.

[Mad97] A. Mader. 'Verification of Modal Properties Using Boolean Equation Systems'. PhD thesis. Technische Universität München, 1997. ISBN: 3-929470-5n-x.

[Mar75] D.A. Martin. 'Borel Determinacy'. In: *The Annals of Mathematics*. Second Series 102.2 (1975), pp. 363–371. ISSN: 0003486X. URL: http://www.jstor.org/stable/1971035.

[Mat03] R. Mateescu. 'A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*. Vol. 2619. LNCS. Springer, 2003, pp. 81–96. DOI: 10.1007/3-540-36577-X_7.

[Mat06] R. Mateescu. 'CAESAR_SOLVE: A generic library for on-the-fly resolution of alternation-free Boolean equation systems'. In: *International Journal on Software Tools for Technology Transfer (STTT)* 8.1 (2006), pp. 37–56. DOI: 10.1007/s10009-005-0194-9.

[MO08a] R. Mateescu and E. Oudot. 'Bisimulator 2.0: An On-the-Fly Equivalence Checker based on Boolean Equation Systems'. In: *2008 6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design*. IEEE, 2008, pp. 73–74. DOI: 10.1109/MEMCOD.2008.4547690.

[MO08b] R. Mateescu and E. Oudot. 'Improved On-the-Fly Equivalence Checking Using Boolean Equation Systems'. In: *Model Checking Software*. Vol. 5156. LNCS. Springer, 2008, pp. 196–213. DOI: 10.1007/978-3-540-85114-1_15.

[McN93] R. McNaughton. 'Infinite games played on finite graphs'. In: *Annals of Pure and Applied Logic* 65.2 (1993), pp. 149–184. DOI: 10.1016/0168-0072(93)90036-D.

[Mil80] A.J.R.G. Milner. *A Calculus of Communicating Systems*. Vol. 92. LNCS. Springer, 1980. DOI: 10.1007/3-540-10235-3.

[Mil89] A.J.R.G. Milner. *Communication and Concurrency*. PHI Series in computer science. Prentice Hall, 1989, pp. I–XI, 1–260. ISBN: 978-0-13-115007-2.

[MPW92a] A.J.R.G. Milner, J. Parrow and D. Walker. 'A Calculus of Mobile Processes, I'. In: *Information and Computation* 100.1 (1992), pp. 1–40. DOI: 10.1016/0890-5401(92)90008-4.

[MPW92b] A.J.R.G. Milner, J. Parrow and D. Walker. 'A Calculus of Mobile Processes, II'. In: *Information and Computation* 100.1 (1992), pp. 41–77. DOI: 10.1016/0890-5401(92)90009-5.

[MJ84] F.L. Morris and C.B. Jones. 'An Early Program Proof by Alan Turing'. In: *IEEE Annals of the History of Computing* 6.2 (1984), pp. 139–143. DOI: 10.1109/MAHC.1984.10017.

[Mos+01] M.W. Moskewicz et al. 'Chaff: engineering an efficient SAT solver'. In: *Design Automation Conference, 2001. Proceedings*. 2001, pp. 530–535. DOI: 10.1109/DAC.2001.156196.

[MB08] L. de Moura and N. Bjørner. 'Z3: An Efficient SMT Solver'. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 4963. LNCS. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[MRG05] M.R. Mousavi, M.A. Reniers and J.F. Groote. 'Notions of Bisimulation and Congruence Formats for SOS with Data'. In: *Information and Computation* 200.1 (2005), pp. 107–147. DOI: 10.1016/j.ic.2005.03.002.

[NV95] R. de Nicola and F.W. Vaandrager. 'Three logics for branching bisimulation'. In: *Journal of the ACM* 42.2 (1995), pp. 458–487. DOI: 10.1145/201019.201032.

[Niw86] D. Niwiński. 'On fixed-point clones'. In: *International Colloquium on Automata, Languages and Programming on Automata, languages and programming*. Vol. 226. LNCS. Springer-Verlag New York, Inc., 1986, pp. 464–473. DOI: 10.1007/3-540-16761-7_96.

[NW83] R. Nowakowski and P. Winkler. 'Vertex-to-vertex pursuit in a graph'. In: *Discrete Mathematics* 43.2-3 (1983), pp. 235–239. DOI: 10.1016/0012-365X(83)90160-7.

[Obd03] J. Obdržálek. 'Fast Mu-Calculus Model Checking when Tree-Width Is Bounded'. In: *Computer Aided Verification*. Vol. 2725. LNCS. Springer, 2003, pp. 80–92. DOI: 10.1007/978-3-540-45069-6_7.

[Obd06a] J. Obdržálek. 'Algorithmic Analysis of Parity Games'. PhD thesis. Laboritory for Foundations of Computer Science, School of Informatics, University of Edinburgh, 2006. URL: http://www.fi.muni.cz/~xobdrzal/papers/thesisObdrzalek.pdf (visited on 19/07/2013).

[Obd06b] J. Obdržálek. 'DAG-width'. In: *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm - SODA '06*. ACM Press, 2006, pp. 814–821. DOI: 10.1145/1109557.1109647.

[OWW09] S. Orzan, J.W. Wesselink and T.A.C. Willemse. 'Static Analysis Techniques for Parameterised Boolean Equation Systems'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*. Vol. 5505. LNCS 642. Springer, 2009, pp. 230–245. DOI: 10.1007/978-3-642-00768-2_22.

[OW10] S. Orzan and T.A.C. Willemse. 'Invariants for Parameterised Boolean Equation Systems'. In: *Theoretical Computer Science* 411.11-13 (2010), pp. 1338–1371. DOI: 10.1016/j.tcs.2009.11.001.

[ORS92] S. Owre, J.M. Rushby and N. Shankar. 'PVS: A Prototype Verification System'. In: *11th International Conference on Automated Deduction (CADE)*. Vol. 607. Lecture Notes in Artificial Intelligence. Springer, 1992, pp. 748–752. URL: http://www.csl.sri.com/papers/cade92-pvs/.

[PT87] R. Paige and R.E. Tarjan. 'Three Partition Refinement Algorithms'. In: *SIAM Journal on Computing* 16.6 (1987), pp. 973–989. DOI: 10.1137/0216062.

[Pan+07] J. Pang et al. 'Model checking a cache coherence protocol of a Java DSM implementation'. In: *The Journal of Logic and Algebraic Programming* 71.1 (2007), pp. 1–43. DOI: 10.1016/j.jlap.2006.08.007.

[Par81] D. Park. 'Concurrency and automata on infinite sequences'. In: *Proc. Theoretical Computer Science*. Vol. 104. LNCS. Springer, 1981, pp. 167–183. DOI: 10.1007/BFb0017309.

[Pau89] L.C. Paulson. 'The foundation of a generic theorem prover'. In: *Journal of Automated Reasoning* 5.3 (1989), pp. 363–397. DOI: 10.1007/BF00248324.

[Pel04] R. Pelánek. 'Typical Structural Properties of State Spaces'. In: *Model Checking Software*. Vol. 2989. LNCS. Springer, 2004, pp. 5–22. DOI: 10.1007/978-3-540-24732-6_2.

[Pel06] R. Pelánek. *Web portal for benchmarking explicit model checkers*. Tech. rep. FIMU-RS-2006-03. Faculty of Informatics Masaryk University Brno, 2006, p. 39. URL: http://www.fi.muni.cz/reports/files/2006/FIMU-RS-2006-03.pdf.

[Pel07] R. Pelánek. 'BEEM: Benchmarks for Explicit Model Checkers'. In: *Model Checking Software*. Vol. 4595. LNCS. Springer, 2007, pp. 263–267. DOI: 10.1007/978-3-540-73370-6_17.

[PV01] V. Petersson and S.G. Vorobyov. 'A Randomized Subexponential Algorithm for Parity Games'. In: *Nordic Journal of Computing* 8.3 (2001), pp. 324–345. URL: http://www.cs.helsinki.fi/njc/References/peterssonv2001:324.html.

[Pet62] C.A. Petri. 'Kommunikation mit Automaten'. PhD thesis. Universität Hamburg, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

[Plo04] G.D. Plotkin. 'A structural approach to operational semantics'. In: *Journal of Logic and Algebraic Programming (JLAP)* 60 (2004), pp. 17–139.

[Pnu77] A. Pnueli. 'The temporal logic of programs'. In: *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*. IEEE, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[PT09] J.C. van de Pol and M. Timmer. 'State Space Reduction of Linear Processes'. In: *Automated Technology for Verification and Analysis 7th International Symposium, ATVA 2009*. Vol. 817. Springer, 2009, pp. 54–68. DOI: 10.1007/978-3-642-04761-9_5.

[PW08] J.C. van de Pol and M. Weber. 'A Multi-Core Solver for Parity Games'. In: *Electronic Notes in Theoretical Computer Science* 220.2 (2008), pp. 19–34. DOI: 10.1016/j.entcs.2008.11.011.

[QS82] J. Queille and J. Sifakis. 'Specification and verification of concurrent systems in CESAR'. In: *International Symposium on Programming*. Vol. 137. LNCS. Springer, 1982, pp. 337–351. DOI: 10.1007/3-540-11494-7_22.

[Qui78] A. Quilliot. 'Jeux et pointes fixes sur les graphes'. PhD thesis. Université de Paris VI, 1978.

[RSW12] M.A. Reniers, R. Schoren and T.A.C. Willemse. 'Results on Embeddings Between State-Based and Event-Based Systems'. In: *The Computer Journal* (2012), pp. 1–20. DOI: 10.1093/comjnl/bxs156.

[RS86] N. Robertson and P.D. Seymour. 'Graph minors. II. Algorithmic aspects of tree-width'. In: *Journal of Algorithms* 7.3 (1986), pp. 309–322. DOI: 10.1016/0196-6774(86)90023-4.

[Rob01] A. Robinson, ed. *Handbook of Automated Reasoning*. Elsevier, 2001, p. 2128. ISBN: 978-0-444-50813-3.

[Ros05] B. Rose. *Othello: A Minute to Learn... A Lifetime to Master*. 2005. URL: http://othellogateway.com/rose/book.pdf.

[Saf88] S. Safra. 'On the complexity of omega-automata'. In: *29th Annual Symposium on Foundations of Computer Science*. IEEE, 1988, pp. 319–327. DOI: 10.1109/SFCS.1988.21948.

[Sch07] S. Schewe. 'Solving Parity Games in Big Steps'. In: vol. 4855. LNCS. Springer, 2007, pp. 449–460. DOI: 10.1007/978-3-540-77050-3.

[Sch08a] S. Schewe. 'An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games'. In: *Computer Science Logic*. Vol. 5213. LNCS. Springer, 2008, pp. 369–384. DOI: 10.1007/978-3-540-87531-4.

[Sch08b] S. Schewe. 'Synthesis of Distributed Systems'. Phd thesis. Universität des Saarlandes, 2008. URL: http://www.csc.liv.ac.uk/~sven/PhD.pdf.

[ST93] P.D. Seymour and R. Thomas. 'Graph Searching and a Min-Max Theorem for Tree-Width'. In: *Journal of Combinatorial Theory, Series B* 58.1 (1993), pp. 22–33. DOI: 10.1006/jctb.1993.1027.

[SLL02] J.G. Siek, L.Q. Lee and A. Lumsdaine. *Boost Graph Library, The: User Guide and Reference Manual*. Addison-Wesley, 2002, p. 352. ISBN: 0-201-72914-8.

[SM98] M. Sighireanu and R. Mateescu. 'Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): An Experiment with E-LOTOS'. In: *STTT* 2.1 (1998), pp. 68–88. DOI: 10.1007/s100090050018.

[SS98] P. Stevens and C. Stirling. 'Practical Model Checking Using Games'. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*. Vol. 1384. LNCS. Springer, 1998, pp. 85–101. DOI: 10.1007/BFb0054166.

[Sti97] C. Stirling. *Bisimulation, Model Checking and Other Games*. Tech. rep. Swansea: University of Wales, 1997. URL: http://homepages.inf.ed.ac.uk/cps/mathfit.pdf (visited on 19/07/2013).

[TK11] F.W. Takes and W.A. Kosters. 'Determining the diameter of small world networks'. In: *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*. ACM Press, 2011, pp. 1191–1196. DOI: 10.1145/2063576.2063748.

[Tar55] A. Tarski. 'A lattice-theoretical fixpoint theorem and its applications'. In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 285–309.

[Tsa+08] Y.K. Tsay et al. 'GOAL Extended: Towards a Research Tool for Omega Automata and Temporal Logic'. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 4963. LNCS. Springer, 2008, pp. 346–350. DOI: 10.1007/978-3-540-78800-3_26.

[Tur37] A.M. Turing. 'On Computable Numbers, with an Application to the Entscheidungsproblem'. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230.

[Tur49] A.M. Turing. 'Checking a Large Routine'. In: *Report of a Conference on High Speed Automatic Calculating machines*. 1949, pp. 67–69. URL: http://www.turingarchive.org/browse.php/B/8.

[vGK99] S.F.M. van Vlijmen, J.F. Groote and J.W.C. Koorn. 'Chapter 5: The Vital Processor Interlocking'. In: *ENTCS* 21.0 (1999), pp. 1–56. DOI: 10.1016/S1571-0661(05)01189-8.

[Vel+01] R. Veldema et al. 'Source-level global optimizations for fine-grain distributed shared memory systems'. In: *ACM SIGPLAN Notices* 36.7 (June 2001), pp. 83–92. DOI: 10.1145/568014.379578.

[VL92] B. Vergauwen and J. Lewi. 'A linear algorithm for solving fixed-point equations on transition systems'. In: *CAAP '92*. Vol. 581. LNCS. Springer, 1992, pp. 322–341. DOI: 10.1007/3-540-55251-0_18.

[VL94] B. Vergauwen and J. Lewi. 'Efficient local correctness checking for single and alternating Boolean equation systems'. In: *Automata, Languages and Programming*. Vol. 820. LNCS. Springer, 1994, pp. 304–315. DOI: 10.1007/3-540-58201-0_77.

[VJ00] J. Vöge and M. Jurdziński. 'A Discrete Strategy Improvement Algorithm for Solving Parity Games'. In: *Computer Aided Verification*. Vol. 1855. LNCS. Springer, 2000, pp. 202–215. DOI: 10.1007/10722167_18.

[Vu07] T.D. Vu. 'Deciding orthogonal bisimulation'. In: *Formal Aspects of Computing* 19.4 (2007), pp. 475–485. DOI: 10.1007/s00165-007-0023-x.

[Wil10] T.A.C. Willemse. 'Consistent Correlations for Parameterised Boolean Equation Systems with Applications in Correctness Proofs for Manipulations'. In: *CONCUR'10: Concurrency Theory*. Vol. 6269. LNCS. Springer, 2010, pp. 584–598. DOI: 10.1007/978-3-642-15375-4_40.

[Wim+06] R. Wimmer et al. 'Sigref—A Symbolic Bisimulation Tool Box'. In: *Automated Technology for Verification and Analysis 4th International Symposium, ATVA 2006*. Vol. 4218. LNCS. Springer, 2006, pp. 477–492. DOI: 10.1007/11901914_35.

[YG04] K. Yorav and O. Grumberg. 'Static Analysis for State-Space Reductions Preserving Temporal Logics'. In: *Formal Methods in System Design* 25.1 (2004), pp. 67–96. DOI: 10.1023/B:FORM.0000033963.55470.9e.

[Zie98] W. Zielonka. 'Infinite games on finitely coloured graphs with applications to automata on infinite trees'. In: *Theoretical Computer Science* 200.1-2 (1998), pp. 135–183. DOI: 10.1016/S0304-3975(98)00009-7.

# Index

# Summary

## Advanced Reduction Techniques for Model Checking

Modern-day software systems are highly complex, and typically consist of a number of interacting components running in parallel. Verification methods aim to prove correctness of such systems. Model checking is a commonly used verification technique, in which a model of the system under investigation is created. Subsequently, it is checked whether this model satisfies certain properties.

Model checking suffers from the infamous state space explosion problem: the number of states in a system grows exponentially in the number of parallel components. Part of the blow-up is also due to the presence of data in descriptions of model checking problems. Typically model checkers construct the full state space, causing the approach to break down already for a small number of parallel components. Properties are then checked on the state space, either directly, or by first combining the state space with a property, leading to, for example, a Boolean equation system or a parity game. The latter step leads to a further blow up.

In this thesis techniques are developed to counter this state space explosion problem in (parameterised) Boolean equation systems and parity games. The main contributions are as follows:

- A structural operational semantics is presented that can equip Boolean equation systems with arbitrary right hand sides with a graph structure, called structure graph. Classical graph structures for analysing Boolean equation systems typically restrict right hand sides to purely conjunctive or disjunctive form, essentially reducing the equation system to a parity game. We use our graphs to study the effects of this restriction on right hand sides, and show that, in the context of bisimulation, this reduction never poses a disadvantage.

- We investigate the reductions that can be achieved using strong bisimulation reduction of structure graphs, and we investigate idempotence identifying bisimulation. We show that, although it indeed identifies idempotence in a restricted subset of equation systems, it does not live up to its name for full-blown structure graphs.

- The insights we gain by studying structure graphs motivate further investigation of weaker equivalences in the setting of parity games. Since the winner in parity games is determined by the infinitely often recurring priorities, intuitively, finite stretches of similar vertices can be compressed. We define the notions of stuttering

equivalence and governed stuttering equivalence, and show that they preserve the winner in a parity game.

- A new set of benchmarks for parity games is developed due to the unavailability of standard benchmarks. These benchmarks subsume the examples that are used for performance evaluation of solving algorithms in the literature. We provide a description of the benchmarks, and we analyse their characteristics.

  The efficacy of our equivalences for reducing parity games is evaluated using this set of benchmarks. It is shown that large reductions are possible, even reducing parity games to a single vertex with a self-loop. Average reductions of about 50% are achieved. Sometimes the technique allows solving parity games which cannot be solved directly; in general, however, the timing results do not show a clear advantage.

- Finally, we move from *a posteriori* reduction of parity games, to static analysis techniques for parameterised Boolean equation systems, that allow the *a priori* reduction of the induced parity games. Essentially, we present a heuristic that performs a live variable analysis. We show that this analysis is more effective at reducing the complexity than existing techniques, *i.e.* the reductions that we obtain are larger. Correctness of our analysis is shown using a generalisation of the equivalences that we introduced for parity games and Boolean equation systems.

# Samenvatting

## Geavanceerde Reductietechnieken voor Model Checking

Moderne computersystemen zijn uiterst complex, en bestaan gewoonlijk uit een aantal componenten die in parallel worden uitgevoerd, en die interactie met elkaar aangaan. Met verificatiemethoden beogen we correctheid van dit soort systemen aan te tonen. Model checking is een veel gebruikte verificatietechniek, waarbij gebruik gemaakt wordt van een model van het systeem dat geverifieerd moet worden. Van dit model wordt nagegaan of het aan de gewenste eigenschappen voldoet.

Een belangrijk probleem bij model checking is de explosie van de toestandsruimte: het aantal toestanden waarin het systeem zich kan bevinden groeit exponentieel in het aantal parallelle componenten. Deels wordt deze explosie in het aantal toestanden ook veroorzaakt door de aanwezigheid van data in de beschrijving van een model check probleem. De meeste model checkers construeren eerst een volledige representatie van de toestandsruimte, waardoor de aanpak al faalt voor systemen die uit slecht enkele parallelle componenten bestaan. Wanneer er wel een representatie gevormd is, worden de eigenschappen hierop geverifieerd, hetzij rechtstreeks, hetzij door eerst de toestandsruimte met de gewenste eigenschap te combineren tot een parity game of een stelsel van Boolse vergelijkingen. Dit combineren leidt tot een extra toename in het aantal toestanden.

In dit proefschrift ontwikkelen we technieken die de explosie van de toestandsruimte tegengaan in stelsels van Boolse vergelijkingen en parity games. De belangrijkste bijdragen zijn als volgt:

- We koppelen een graaf structuur aan een stelsel van Boolse vergelijkingen met behulp van een operationele semantiek. Bestaande graaf structuren die gebruikt worden in de analyse van dit soort stelsels van vergelijkingen beperken zich tot een subset waarin elke rechterkant van een vergelijking puur conjunctief danwel puur disjunctief is. We gebruiken onze graafstructuur om de gevolgen van deze beperkingen te bestuderen, en we tonen aan dat, in de context van bisimulatie, deze beperking nooit nadelig werkt.

- We onderzoeken de reducties van stelsels van Boolse vergelijkingen door middel van sterke bisimulatie reductie van de structuurgrafen, en we introduceren de zwakkere notie van idempotentie identificerende bisimulatie. We laten zien dat deze verzwakking idempotentie slechts in een subset van stelsels van vergelijkin-

gen identificeert, maar dat het voor structuurgrafen in algemene zin zijn naam niet verdient.

- De inzichten die we verkrijgen door het bestuderen van deze structuurgrafen geven aanleiding tot het verder bestuderen van zwakkere equivalenties in parity games, welke gezien kunnen worden als een subset van deze structuurgrafen. Aangezien de winnaar in dit soort spellen bepaald wordt door de prioriteiten die oneindig vaak voorkomen kunnen we eindige sequenties van vergelijkbare knopen samennemen tot een enkele knoop. We definieren de noties van stuttering equivalentie en governed stuttering equivalentie, en we tonen aan dat deze de winnaar in een spel behouden.

- Bij gebrek aan een standaard set van voorbeelden, ontwikkelen we een benchmark set voor parity game algorithmen, die de spellen in bestaande experimenten uit de literatuur omvat. We beschrijven de invoer van deze experimenten, en bestuderen de karakteristieken.

  De effectiviteit van onze equivalenties wordt bestudeerd aan de hand van deze set van voorbeelden. Onze experimenten tonen aan dat met de technieken die we beschrijven grote reducties haalbaar zijn. Sommige spellen reduceren tot een enkele knoop met een lus, en gemiddeld genomen worden reducties van ongeveer 50% behaald. Soms leiden onze technieken er toe dat spellen die voorheen niet rechtstreeks opgelost konden worden na reductie wel op te lossen zijn. Echter, over het algemeen leiden de technieken niet tot een duidelijke versnelling in het oplossen van parity games.

- Tot slot bekijken we, in plaats van de *a posteriori* reductie van parity games, statische analyse technieken voor stelsels van Boolse vergelijkingen die geparameteriseerd zijn met data. Deze technieken stellen ons in staat om de parity games die uit deze stelsels gegenereerd worden *a priori* te reduceren. Onze techniek is, in essentie, een heuristiek die ongebruikte variabelen opspoort. We laten zien dat deze analyse tot een grotere reductie leidt dan bestaande technieken. De correctheid van de statische analyse wordt aangetoond met behulp van een generalisatie van de equivalenties die we eerder gedefinieerd hebben op parity games en stelsels van Boolse vergelijkingen.

# Curriculum Vitae

Jeroen Johan Anna Keiren was born on 21 July 1986 in Venray and raised in Lottum, The Netherlands. After completing his secondary education, he studied Computer Science and Engineering at Eindhoven University of Technology. He received his Master of Science degree with honours in August 2009. His Master's thesis was titled "An Experimental Study of Algorithms and Optimisations for Parity Games, with an Application to Boolean Equation Systems".

Immediately after his graduation Jeroen started as a PhD candidate at the same university. His project was supervised by prof. dr. ir. Jan Friso Groote and dr. ir. Tim Willemse. Within the project, Jeroen focussed on reduction techniques in formal verification. He also applied model checking techniques to industrial case studies. His scientific contributions in these areas revolve around reduction techniques for parity games and (parameterised) Boolean equation systems. Jeroen also contributed to the mCRL2 toolset for the analysis of systems by specifying fundamental concepts withing the toolset, developing code, improving the documentation, and enhancing the test process.

As of October 2013 he starts working as a university researcher in the Department of Computer Sciences of the Faculty of Sciences at the Vrije Universiteit Amsterdam.

## Titles in the IPA Dissertation Series since 2007

**H.A. de Jong**. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

**N.K. Kavaldjiev**. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

**M. van Veelen**. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

**T.D. Vu**. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

**L. Brandán Briones**. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

**I. Loeb**. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

**M.W.A. Streppel**. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

**N. Trčka**. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

**R. Brinkman**. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

**A. van Weelden**. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

**J.A.R. Noppen**. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

**R. Boumen**. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

**A.J. Wijs**. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

**C.F.J. Lange**. *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML*. Faculty of Mathematics and Computer Science, TU/e. 2007-14

**T. van der Storm**. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

**B.S. Graaf**. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

**A.H.J. Mathijssen**. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

**D. Jarnikov**. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

**M. A. Abam**. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

**W. Pieters**. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

**A.L. de Groot**. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

**M. Bruntink**. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

**A.M. Marin**. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

**N.C.W.M. Braspenning**. *Model-based Integration and Testing of High-tech Multidisciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

**M. Bravenboer**. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

**M. Torabi Dashti**. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

**I.S.M. de Jong**. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

**I. Hasuo**. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

**L.G.W.A. Cleophas**. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

**I.S. Zapreev**. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

**M. Farshi**. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

**G. Gulesir**. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

**F.D. Garcia**. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

**P. E. A. Dürr**. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

**E.M. Bortnik**. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

**R.H. Mak**. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

**M. van der Horst**. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

**C.M. Gray**. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

**J.R. Calamé**. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

**E. Mumford**. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

**E.H. de Graaf**. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

**R. Brijder**. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

**A. Koprowski**. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

**U. Khadim**. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

**J. Markovski**. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

**H. Kastenberg**. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

**I.R. Buhan**. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

**R.S. Marin-Perianu**. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

**M.H.G. Verhoef**. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

**M. de Mol**. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

**M. Lormans**. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

**M.P.W.J. van Osch**. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

**H. Sozer**. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

**M.J. van Weerdenburg**. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

**H.H. Hansen**. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

**A. Mesbah**. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

**A.L. Rodriguez Yakushev**. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

**K.R. Olmos Joffré**. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

**J.A.G.M. van den Berg**. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

**M.G. Khatib**. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

**S.G.M. Cornelissen**. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

**D. Bolzoni**. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

**H.L. Jonker**. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

**M.R. Czenko**. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

**T. Chen**. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

**C. Kaliszyk**. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

**R.S.S. O'Connor**. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

**B. Ploeger**. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

**T. Han**. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

**R. Li**. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

**J.H.P. Kwisthout**. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

**T.K. Cocx**. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

**A.I. Baars**. *Embedded Compilers*. Faculty of Science, UU. 2009-25

**M.A.C. Dekker**. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

**J.F.J. Laros**. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

**C.J. Boogerd**. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

**M.R. Neuhäußer**. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

**J. Endrullis**. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

**T. Staijen**. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

**Y. Wang**. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

**J.K. Berendsen**. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

**A. Nugroho**. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

**A. Silva**. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

**J.S. de Bruin**. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

**D. Costa**. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

**M.M. Jaghoori**. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

**R. Bakhshi**. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

**B.J. Arnoldus**. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

**E. Zambon**. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

**L. Astefanoaei**. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

**J. Proença**. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

**A. Morali**. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

**M. van der Bijl**. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

**C. Krause**. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

**M.E. Andrés**. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

**M. Atif**. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

**P.J.A. van Tilburg**. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

**Z. Protic**. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

**S. Georgievska**. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

**S. Malakuti**. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

**M. Raffelsieper**. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

**C.P. Tsirogiannis**. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

**Y.-J. Moon**. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

**R. Middelkoop**. *Capturing and Exploiting Abstract Views of States in OO Verification*.

Faculty of Mathematics and Computer Science, TU/e. 2011-18

**M.F. van Amstel**. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

**A.N. Tamalet**. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

**H.J.S. Basten**. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

**M. Izadi**. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

**L.C.L. Kats**. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

**S. Kemper**. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

**J. Wang**. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

**A. Khosravi**. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

**A. Middelkoop**. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

**Z. Hemel**. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

**T. Dimkov**. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

**S. Sedghi**. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

**F. Heidarian Dehkordi**. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

**K. Verbeek**. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

**D.E. Nadales Agut**. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

**H. Rahmani**. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

**S.D. Vermolen**. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

**L.J.P. Engelen**. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

**F.P.M. Stappers**. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

**W. Heijstek**. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13

**C. Kop**. *Higher Order Termination*. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

**A. Osaiweran**. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

**W. Kuijper**. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

**H. Beohar**. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

**G. Igna**. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

**E. Zambon**. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

**B. Lijnse**. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

**G.T. de Koning Gans**. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

**M.S. Greiler**. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

**L.E. Mamane**. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

**M.M.H.P. van den Heuvel**. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

**J. Businge**. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

**S. van der Burg**. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

**J.J.A. Keiren**. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11