An experimental study of algorithms and optimisations for parity games, with an application to Boolean Equation Systems

Keiren, J.J.A.

Award date:
2009

An experimental study of algorithms and optimisations for parity games, with an application to Boolean Equation Systems

Jeroen Keiren

Eindhoven, 2nd July 2009

Abstract

We present an empirical study comparing algorithms for solving parity games. The problems that we solve all stem from the problem of model checking modal μ-calculus formulae against realistic specifications. We investigate the practical use of various optimisation techniques for solving parity games, showing that decomposition into strongly connected components and applying efficient algorithms for special cases are highly beneficial. Furthermore, it is shown that the theoretic dependency on the number of priorities manifests itself in practice.

We compare eleven algorithms for solving parity games on a number of concrete model checking examples. It is shown that in general the recursive algorithm due to Zielonka and the bigstep algorithm due to Schewe perform best in practice.

We also relate the problems of finding a winning strategy in parity games with the problem of solving Boolean Equation Systems (BESs), and a way to employ the parity game solving algorithms directly to BESs. Here we also introduce new optimisation techniques for BESs, based on the known theory for parity games. Additionally we demonstrate that there is room for generalisation of parity game algorithms when applied in the BES framework by generalising the small progress measures algorithm due to Jurdziński. This can serve as a basis for further investigation of the generalisation of the efficient bigstep algorithm in the BES framework, as this algorithm uses small progress measures for solving sub-problems.

Contents

Chapter 1

Introduction

Software and hardware systems are getting more and more complex as the capacity of modern day computer systems increases. Correctness of this complex software and hardware is not evident, hence we require techniques for convincing ourselves of this correctness. Simulation and testing are much used devices for validation of designs of such systems. The problem with these techniques however, is that the effectiveness drops when the number of bugs present decreases. In fact, the absence of bugs cannot be shown at all with these approaches.

In order to shown that a system is bug-free, more powerful techniques are needed. This is where formal verification comes into play. One way of verifying a system is using proof techniques, *i.e.* employ mathematical reasoning to verify a system. Usually this is done using theorem provers such as HOL [GA94], Isabelle [Pau86], PVS [ORS92] and Coq [BC04]. The major drawback of this approach is that human intellect is required in the verification process and verification is very time consuming.

Model checking on the other hand performs an exhaustive exploration of the entire state space of a model of a system in a fully automatic way. This eliminates the requirements of human intellect in the verification process. On the other hand model checking suffers from the well-known state explosion problem. When a system consists of a number of parallel operating components the number of states of the entire system can grow exponentially. The state explosion problem is combated by the development of symbolic verification techniques, as well as techniques for reducing the state space prior to performing the actual model checking.

In model checking a desired property is expressed as a logical formula and it is checked whether a model of the system satisfies this formula. Various logics are used in this area, *e.g.* CTL [CES86] and LTL [Pnu77]. In this thesis we are concerned with an extension of the modal μ-calculus due to Kozen [Koz83], which subsumes many of the other logics.

Generally, model checking procedures for modal μ-calculus can be split into two categories: local and global procedures. Local procedures are used to show that a certain state in the system satisfies a requirement, whereas global procedures compute for all states whether they satisfy a requirement. For both approaches algorithms have been presented in the literature. For the full fragment of modal μ-calculus efficient algorithms are not known, and the problem is known to be in $NP \cap co\text{-}NP$ as well as in $UP \cap co\text{-}UP$. As it is generally believed that NP and $co\text{-}NP$ are unequal (meaning that no problem can be in both NP and $co\text{-}NP$), it is believed that a polynomial time algorithms can be found, but no such algorithm is known at present. Known methods for solving the model checking problem for the modal μ-calculus include BDD based methods using iteration for fixpoint computation [EL86], translation to the problem of finding a winning strategy in a parity game [Sti95, Sti96], finding a switch setting in switching graphs [GP08], as well as translating the problem to finding solutions for a Boolean Equation System [Mad97].

Our main interest lies in the symbolic approach to model checking of modal μ-calculus through Parameterized Boolean Equation Systems (PBESs) [GW05], and the reduction thereof to Boolean Equation Systems (BESs) as used in *e.g.* the mCRL2 toolset [GMWU07].

The described theory for BESs and PBESs is still developing quickly. Various algorithms for

solving BESs are in the literature, *e.g.* approximation, tableaux based algorithms and Gauß elimination [Mad97]. All these algorithms are exponential in the size of the BES. For some special cases more efficient algorithms are known [GK05, Mat03].

Our experience shows that the generic algorithms for solving BESs are not sufficiently efficient in practice. In this thesis we investigate other algorithms and techniques that improve the performance of solving Boolean Equation Systems.

It is known that BESs and parity games are closely related, see *e.g.* [Kei06]. In fact parity games coincide with the subset of BESs in which conjuncts and disjuncts do not occur mixed in an equation, and the Boolean constants true and false do not occur (Simple Recursive Form). Because of this, algorithms for solving parity games can be employed to solve this subset of BESs. Furthermore each BES can be transformed to this subset in linear time, and at the cost of a linear blow-up in the size of the BES.

Finding efficient algorithms for parity games is a popular field in current research. This is witnessed by the large number of algorithms [Lan05, FL09, McN93, Zie98, VJ00, SV00, Sch08, Jur00, JPZ06, SS98, Sch07] of different worst-case running time complexities.

In addition, meta-level simplification techniques have been developed, that can be used to speed up all of these algorithms, see *e.g.* [FL09].

We classify these meta-level (simplification) techniques from the parity game framework, and investigate their BES counterparts, introducing new theory in the BES framework. Separately we have developed a bisimulation like technique for reducing BESs, which we will introduce briefly in this thesis, of which the full details with experiments can be found in [KW09]. Furthermore we investigate how the algorithms for solving parity games perform for realistic model checking problems. The analysis of over 20,000 runs, worth about 3 months of CPU time, provides us with insight as to which algorithms and optimisations perform well in practice, and hence should be considered for translation to the realm of BESs. The expectation is that based on our results one or two of the best performing algorithms will be implemented in the mCRL2 toolset. Finally we give a translation from a parity game algorithm to the BES framework as a proof of concept.

Outline The rest of this thesis is structured as follows. Chapter 2 introduces model checking of modal μ-calculus. In Chapter 3 parity games and BESs are introduced. Chapter 4 introduces the algorithms for solving parity games that stem from literature. In Chapter 5 these algorithms are evaluated through extensive experimental comparisons. Chapter 6 shows an example of a parity game algorithm applied directly to BES in simple recursive form, as well as a generalisation of the algorithm to BES with arbitrary right hand sides. Finally Chapter 7 summarises the conclusions that follow from the earlier chapters.

Chapter 2

Context

In this chapter we introduce, on an abstract level, the context in which this work is carried out. The formalisms of LPE, modal μ-calculus and Parameterized Boolean Equation System (PBES) are briefly introduced. Parity games and Boolean Equation Systems (BESs) will be treated more in depth in the next chapter.

2.1 Process description

A Labelled Transition System (LTS) is a structure $\langle \Sigma, \rightarrow \rangle$ where Σ is a set of states, and $\rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation. Transitions are labelled with actions. A LTS describes the states that can be reached in a process, and the transitions that can be made.

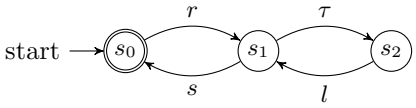
In this thesis we consider model checking problems in the symbolic setting of Linear Process Equations (LPE, a process algebraic, symbolic encoding of a Labelled Transition System). An LPE basically describes a process using a set of condition, action, effect rules that modify some global state. For an in-depth treatment of LPEs we refer the reader to *e.g.* [Use02].

2.2 Modal μ-calculus

The modal μ-calculus is a powerful logic for expressing properties of concurrent systems. The modal μ-calculus is a modal logic that features extremal fixpoints. It was originally introduced by Scott and De Bakker[1], and introduced in its most used form by Kozen [Koz83]. We consider an extension of the modal μ-calculus with data variables, quantifiers and parametrisation as described in the literature, see *e.g.* [GW05], allowing for the expression of data dependent properties.

As an example of a model checking problem consider a μ-calculus model checking problem, taken from [RW09], involving an unreliable channel.

Example 2.2.1. The channel can read messages from the environment, and send or lose these next. 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.



We use the formula that expresses 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. The problem is formalised as follows:

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

[1] According to [Koz83, page 333]

Using the translation of Mader [Mad97], the BES given below is obtained. The solution to X_{s_i} answers whether s_i satisfies formula ϕ.

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

Note that the solution to all propositional variables occurring in the BES is true, hence the property holds in all states.

In addition to the syntax from our example, the μ-calculus that we use allows for the expression of universal and existential quantifications over data.

2.3 Parameterized Boolean Equation Systems

Parameterized Boolean Equation Systems (PBESs) are generalisations of Boolean Equation Systems. A PBES is a sequence of equations of the form $\sigma X(d_1{:}D_1, \ldots, d_n{:}D_n) = \phi$, where $\sigma \in \{\mu, \nu\}$ is a fixpoint symbol, d_i is a data variable of sort D_i and ϕ is a predicate formula. PBESs were introduced by Groote and Mateescu [Mat97, GM99] as an intermediate formalism for model checking process with arbitrary data, also allowing for the encoding the verification problem for systems with infinite state spaces.

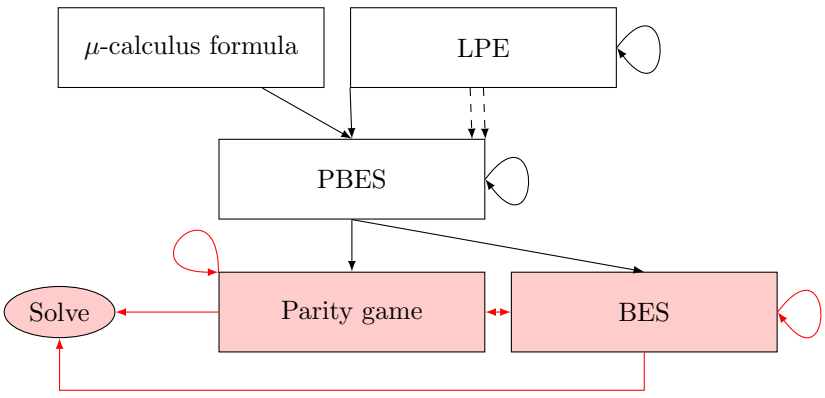
2.4 Integration

In Figure 2.1 we show an overview of the methodology we use. A PBES is obtained either by combining a modal μ-calculus formula and an LPE using the translation in [GM99], or by combining two LPEs into an PBES encoding a process equivalence using [CPPW07]. The PBES is of the form discussed in the previous section.

A PBES can be transformed into a Boolean Equation System using the methods described in [vDPW]. BESs and parity games are equivalent, as we will show in Section 3.3. This is also shown in the figure by a conversion between the two. Because of the equivalence of BES and parity games the same methods can be applied to obtain a parity game from a PBES. The BES and parity games can consequently be solved.

At a number of levels intermediate transformations can be applied to simplify the given structure, these are denoted with self-loops at the LPE, PBES, parity game and BES levels.

Figure 2.1: Overview of our methodology



In this thesis we concentrate on transformations on parity games and BESs, the equivalence between both frameworks, and algorithms for solving parity games and BESs. These parts are highlighted in Figure 2.1. BESs and parity games will be treated in-depth in the next chapter.

Chapter 3

Parity games and Boolean Equation Systems

This chapter introduces parity games and Boolean Equation Systems in detail. A thorough understanding is developed with respect to the commonalities between both frameworks. This gives rise to new insights, especially in the BES framework.

3.1 Parity games

A parity game is a graph game played by two players, *Even* and *Odd* on a game graph in which each vertex is assigned an integer priority. An infinite play is an infinite path in the graph in which a player does a step if a token is on a vertex for that player. Player *Even* wins an infinite play if the lowest priority that occurs infinitely often in that play is even, otherwise player *Odd* wins the play. We use the generic *Player* to denote either *Even* or *Odd* in case definitions are analogous for both players. We use the convention that $\overline{Even}$ equals *Odd* and $\overline{Odd}$ equals *Even* and $\overline{Player}$ is used in the same way.

A game graph is a directed graph $\mathcal{G} = (V, E, p)$, in which V is a set of vertices, $E \subseteq V \times V$ is a total edge relation, *i.e.* for each $v \in V$ there is a $w \in V$ such that $(v, w) \in E$, and $p{:}V \to \mathbb{N}$ is a priority function, assigning a non-negative integer priority to each vertex. We restrict ourselves to games on finite graphs.

Definition 3.1.1. Given game graph $\mathcal{G} = (V, E, p)$, and partition (V_{Even}, V_{Odd}) of V, $\Gamma = (V, E, p, (V_{Even}, V_{Odd}))$ is a parity game.

Instead of $(v, w) \in E$ we also write infix notation vEw. Furthermore we write the set of successors of v as $vE \triangleq \{w \mid vEw\}$, and the set of predecessors of w as $Ew \triangleq \{v \mid vEw\}$.

A parity game is played by the two players by placing a token on an initial vertex—this vertex may be chosen arbitrarily, but in practice the choice corresponds to the model checking problem that we are trying to solve, *i.e.* depending on the state in the model for which we want to know the answer to the model checking problem a choice is made for an initial vertex. Moves are taken indefinitely according to the following simple rule: if the token is on a vertex $v \in V_{Player}$ then *Player* moves the token along an outgoing edge of v. The result is an infinite path (also referred to as a play) $\pi = \langle v_1, v_2, v_3, \dots \rangle$ in the game graph.

Let $Inf(\pi)$ denote the set of priorities occurring infinitely often in play π. Play π is winning for player *Even* if and only if $\min(Inf(\pi))$ is even, π is winning for player *Odd* otherwise.[1]

A strategy for *Player* is a partial function $\psi_{Player}{:}V^*V_{Player} \to V$ that decides the vertex the token is played to based on the history of vertices that has been visited (V^* are the vertices in the history, V_{Player} represents the current vertex).

[1] Note that this is the typical definition for min-parity games, for the notion of max-parity games, replace $\min(Inf(\pi))$ with $\max(Inf(\pi))$.

A play $\pi = \langle v_1, v_2, v_3, \dots \rangle$ is consistent with a strategy ψ_{Player} for *Player* if and only if every vertex $v_i \in \pi$ is such that $v_i \in V_{Player}$ is immediately followed by $v_{i+1} = \psi_{Player}(\langle v_1, \dots, v_i \rangle)$

Definition 3.1.2. Strategy ψ_{Player} is a winning strategy for *Player* from set $W \subseteq V$ if every play starting from a vertex in W, consistent with ψ_{Player} is winning for *Player*.

It is known that a player has a winning strategy for a game if and only if the opponent does not. This property is referred to as determinacy.

Theorem 3.1.3. [Mar75, GH82, EJ91, McN93] For every parity game, there is a unique partition (W_{Even}, W_{Odd}) of V such that there is a winning strategy ψ_{Even} for player *Even* from his winning set W_{Even} and a winning strategy ψ_{Odd} for player *Odd* from her winning set W_{Odd}.

We refer to finding the partition (W_{Even}, W_{Odd}) as solving the parity game.

For finding winning strategies, and hence the division in W_{Even} and W_{Odd}, it suffices to look at history free strategies [EJ91, McN93, Zie98] (also referred to as positional or memory-less strategies). In a history free strategy a given vertex v_i always gets the same successor v_{i+1}, regardless of the path by which v_i is reached. We define such a strategy for a player, fixing an outgoing edge for each vertex in the set corresponding to that player.

Definition 3.1.4. [EJ91, McN93, Zie98] A function $\psi_{Player}{:}V_{Player} \to V$ is a history-free strategy for player *Player* iff $(v, \psi(v)) \in E$ for all $v \in V_{Player}$.

Consistency with a history-free strategy is defined similarly to consistency for arbitrary strategies. In the sequel we restrict ourselves to such history free strategies.

Now that we have established a basic understanding of parity games we introduce some derived notions that are used in the algorithms that we investigate.

A set $U \subseteq V$ is *Player*-closed [FL09] if and only if *Player* can force the game to stay within U from any vertex $v \in U$. That is, $\forall v \in (U \cap V_{\overline{Player}}) : vE \subseteq U$ and $\forall v \in (U \cap V_{Player}) : vE \cap U \neq \emptyset$. So $\overline{Player}$ must not be able to leave U, whereas *Player* must have the possibility to stay within U.

Given a history-free strategy ψ_{Player} and a parity game $\Gamma = (V, E, p{:}V \to \mathbb{N}, (V_{Even}, V_{Odd}))$, the subgame $\Gamma \mid_\psi$ is defined as $(V, E \mid_\psi, p, (V_{Even}, V_{Odd}))$, where

$$E \mid_\psi \stackrel{\Delta}{=} E \setminus \{(u, v) \in E \mid u \in V_{Player} \wedge v \neq \psi_{Player}(u)\}$$

In other words, $\Gamma \mid_\psi$ is the same game as Γ, except that from the vertices of *Player* the choice is dictated by the strategy for *Player*.

Similarly we define the subgame $\Gamma \mid_U$ which is induced by set $U \subseteq V$ as

$$\Gamma \mid_U \stackrel{\Delta}{=} (U, E \cap U^2, p \mid_U, (V_{Even} \cap U, V_{Odd} \cap U))$$

where $p \mid_U (v) = p(v)$ if $v \in U$, and is undefined otherwise. Note that a subgame is only induced if edge relation $E \cap U^2$ remains total. Each *Player*-closed set U induces a subgame [FL09].

Definition 3.1.5. We call a path $\pi = \langle v_1, \dots, v_n \rangle^\omega$ in a parity game an i-cycle if and only if the lowest priority occurring the path is i, i.e. $i = \min(\{p(v_j) \mid 1 \leqslant j \leqslant n\})$.

We refer to an i-cycle with even i as an even cycle, similarly an i-cycle with odd i is referred to as an odd cycle.

A subgame $\Gamma \mid_U$ that is won by *Player* by forcing player $\overline{Player}$ to stay in U, using a winning strategy on U is called a *Player*-dominion [FL09] (also referred to as $\overline{Player}$-trap [Zie98]). We define a *Player*-dominion formally as follows:

Definition 3.1.6. [Zie98, FL09] Let $\Gamma = (V, E, p, (V_{Even}, V_{Odd}))$ be a parity game. Set $U \subseteq V$ is a *Player*-dominion if and only if U is *Player*-closed and the subgame $\Gamma \mid_U$ is won by player *Player*.

We also define the attractor set for *Player* for a set $U \subseteq V$. This is the set of vertices such that *Player* can force any play to reach U.

Definition 3.1.7. [McN93, Zie98, FL09] Let $U \subseteq V$. We define the attractor sets inductively as follows:

$$\begin{aligned}
Attractor^0_{Player}(U) &= U \\
Attractor^{k+1}_{Player}(U) &= Attractor^k_{Player}(U) \\
&\cup \ (V_{Player} \cap \{v \mid vE \cap Attractor^k_{Player}(U) \neq \emptyset\}) \\
&\cup \ (V_{\overline{Player}} \cap \{v \mid vE \subseteq Attractor^k_{Player}(U)\}) \\
Attractor_{Player}(U) &= \bigcup_{k \in \mathbb{N}} Attractor^k_{Player}(U)
\end{aligned}$$

Decomposition of the game according to the dominions of the players can be used to speed up computation of the solution of the entire game.

Lemma 3.1.8. [McN93, Sti95, Zie98] Let $\Gamma = (V, E, p{:}V \to \mathbb{N}, (V_{Even}, V_{Odd}))$ be a parity game, and $U \subseteq V$. The edge relation $E \cap V^2$ in the game $\Gamma\,|_U$ is total.

Note that $V \setminus Attractor_{Player}(U)$ is $\overline{Player}$-closed [Zie98], *i.e.* *Player* can force the game to stay in $V \setminus Attractor_{Player}(U)$. If this were not the case, then there would be a vertex $v \in V_{Player}$, from which a node in $Attractor_{Player}(U)$ can be reached, but then it would be part of $Attractor_{Player}(U)$ according to the definition of *Attractor*. Furthermore, if U is a *Player*-dominion, then $Attractor_{Player}(U)$ is also a *Player*-dominion.

In the literature several algorithms for solving parity games have been proposed, all of which are worst-case exponential. We will discuss these algorithms in more detail in Chapter 4.

3.1.1 Simplifications of parity games

In the literature some simplifications of parity games are used that work independently of the algorithm that is used for solving a parity game. They are also implemented in actual tools, *e.g.* the PGSolver toolset [FL09]. We give brief descriptions of these optimisations here. Correctness arguments can be found in [FL09]. Throughout this section, assume a given, arbitrary parity game $\Gamma = (V, E, p, (V_{Even}, V_{Odd}))$.

Edge reducing optimisations

All optimisations in this section reduce the number of edges in the parity game, hence it speeds up computation by algorithms that depend on the number of edges.

Self-cycle parity games Consider a vertex v in Γ, such that vEv, i.e. v has a self-loop. If $p(v)$ is even, and $v \in V_{Even}$, or $p(v)$ is odd and $v \in V_{Odd}$ then $\{v\}$ is a *Player*-dominion, hence its attractor can be removed (see SCC decomposition). If $p(v)$ is odd and $v \in V_{Even}$, or $p(v)$ is even and $v \in V_{Odd}$ then taking the self-loop is always bad for *Player*, hence it can be removed from the game, as long as totality is preserved. Evidently this optimisation reduces the number of edges in the game, and hence may speed up computation of the solution.

Priority reducing optimisations

The optimisations in this section reduce the number of priorities in the parity game. As most algorithms depend exponentially on the number of priorities, this might greatly affect the time required for computing the solution to a parity game.

Priority compaction Observe that for computing winning sets and strategies the actual priorities are not relevant. Instead only their ordering and parity is of importance. Using this observation we can use the following reduction. Let $\equiv_2$ denote equivalence modulo 2. Consider two priorities $p_1 < p_2$ in a game, and $p_1 \equiv_2 p_2$, but there is no p' such that $p_1 < p' < p_2$ and $p' \not\equiv_2 p_1$, then each priority p_2 can be replaced by p_1, preserving the winning sets and strategies of the original game. If it is assumed that there always is some vertex with priority 0, priority compaction results in a game in which the least priority is either 0 or 1 (1 if the lowest priority in the original game was odd, 0 otherwise). This optimisation reduces the number of priorities that occurs in the parity game.

Priority propagation Another optimisation that reduces the number of priorities in the game is priority propagation. Note that this does in general not preserve the parities of nodes, *i.e.* a node with even priority may change into a node with odd priority and the other way round. Observe that any play that visits some vertex v infinitely often must also visit one of v's successors infinitely often. If the priorities $p(u)$ of all successors u of v are less than the priority $p(v)$ of v, then we can replace $p(v)$ with $p(u)$ as $p(v)$ is certainly not the smallest priority on the play.[2] The same holds for the predecessors of v. Like priority compaction, this optimisation reduces the number of priorities that occurs in the parity game.

Algorithms for special cases

The algorithms presented in this section solve special cases of parity games, where the game either involves a single player or a single parity. These algorithms are more efficient than the more generic algorithms.

Single parity parity games If all nodes v in a strongly connected component have the same priority $p(v)$, then the whole game in the SCC is won by player *Even* if $p(v)$ is even, *Odd* if $p(v)$ is odd. A winning strategy can be computed by random choice. This optimisation is beneficial as it computes a solution for a parity game in time linear in the number of edges.

Single player parity games Γ is a single-player parity game for *Player* if and only if all $v \in V_{\overline{Player}}$ have exactly one outgoing edge (i.e. $\mid vE \mid = 1$). Fixed-point iteration can be used to solve a one-player game that is a single SCC. *Player* wins the game if there is a node u of which the priority $p(u)$ corresponds with the player owning u, and u is reachable from itself on a path that does not contain vertices w with $p(w) < p(u)$.[3] If there is such a cycle then the rest of the SCC lies in the attractor of the cycle, hence *Player* wins the subgame in the SCC. If there is no such cycle, then the whole subgame in the SCC is won by $\overline{Player}$. The advantage of this optimisation is that a solution can be computed independent of the number of priorities that occur in the parity game, hence abstracting from that aspect of the running times of the algorithms.

SCC decomposition

On a more general level, decomposition into strongly connected components can be used to speed up computation of the solution for a parity game. Generic parity game algorithms, combined with the aforementioned optimisation techniques, can be applied per strongly connected component as follows.

A strongly connected component (SCC) is a maximal non-empty subset $S \subseteq V$ such that each vertex in S can reach each other vertex in S. Given that the game graph underlying the parity game is a directed graph, the graph obtained by contracting each SCC into a single vertex (the graph of strongly connected components) is a directed acyclic graph (DAG).

[2] For max-parity games, the priorities of all successors u of v need to be greater than the priority of v, and we take the greatest of these priorities

[3] $p(w) > p(u)$ in case of max-parity games

We can optimise solving parity games using SCC decomposition as follows. First decompose the parity game into SCCs. All terminal SCCs (SCCs that do not have outgoing edges in the resulting DAG), including trivial terminal SCCs, are solved using a parity game solver. Observe that these local solutions can be used as solutions in the global game.

Next the attractors for both players are computed with respect to their winning sets obtained from the solved SCCs. These attractors are removed from the game. The result is still a game, but some of the SCCs may not be SCCs anymore (because of the removal of vertices in the attractor sets). These modified SCCs are again decomposed into SCCs resulting in a new decomposition. This process is repeated until the entire game has been solved.

This results in subgames that are smaller with respect to the number of vertices and the number of edges. In most cases it will also be the case that the number of priorities in such a subgame is smaller than the number of priorities in the entire game. Furthermore it may give rise to single player or single parity games which can in turn be solved efficiently.

3.2 Boolean Equation Systems

Boolean Equation Systems (BESs) [Mad97] are a class of equation system that can be employed to perform model checking of modal μ-calculus formulae. Basically, BESs are finite sequences of least and greatest fixpoint equations, where each right-hand side of an equation is a proposition in positive form. It has been shown [Mad97] that solving a BES is equivalent to the model-checking problem. BESs are used for this purpose in *e.g.* the tool sets CADP [GLMS07] and mCRL2 [GMWU07]. Several algorithms for solving BESs exist, see [Mad97, Kei06]. Furthermore there are efficient algorithms for some special cases, see [Mat03, Kei06]. We formally introduce the theory required for understanding the results obtained in this paper.

Definition 3.2.1. We assume a set $\mathcal{X}$ of Boolean variables, with typical elements X, X_1, X_2, ... and a type $\mathbb{B}$ with elements true, false representing the Booleans. Furthermore we have fixpoint symbols μ for least fixpoint and ν for greatest fixpoint.

A Boolean Equation System is a system of fixpoint equations, inductively defined as follows:

- ϵ is the empty BES
- if $\mathcal{E}$ is a BES, then $(\sigma X = f)\mathcal{E}$ is also a BES, with $\sigma \in \{\mu, \nu\}$ a fixpoint symbol and f a negation free formula over $\mathcal{X}$, defined by the following grammar:

$$f, g ::= c \mid X \mid f \wedge g \mid f \vee g$$

  where $X \in \mathcal{X}$ is a proposition variable of type $\mathbb{B}$ and $c \in \{\mathsf{true}, \mathsf{false}\}$ is a Boolean constant.

For any equation system $\mathcal{E}$, the set of *bound proposition variables*, $\mathsf{bnd}(\mathcal{E})$, is the set of variables occurring at the left-hand side of some equation in $\mathcal{E}$. The set of *occurring proposition variables*, $\mathsf{occ}(\mathcal{E})$, is the set of variables occurring at the right-hand side of some equation in $\mathcal{E}$.

$$\mathsf{bnd}(\epsilon) \triangleq \emptyset \qquad \mathsf{bnd}((\sigma X = f)\ \mathcal{E}) \triangleq \mathsf{bnd}(\mathcal{E}) \cup \{X\}$$
$$\mathsf{occ}(\epsilon) \triangleq \emptyset \qquad \mathsf{occ}((\sigma X = f)\ \mathcal{E}) \triangleq \mathsf{occ}(\mathcal{E}) \cup \mathsf{occ}(f)$$

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

$$\mathsf{occ}(c) \triangleq \emptyset \qquad \mathsf{occ}(X) \triangleq \{X\}$$
$$\mathsf{occ}(f \vee g) \triangleq \mathsf{occ}(f) \cup \mathsf{occ}(g) \qquad \mathsf{occ}(f \wedge g) \triangleq \mathsf{occ}(f) \cup \mathsf{occ}(g)$$

BESs $\mathcal{E}$ and $\mathcal{F}$ with $\mathsf{bnd}(\mathcal{E}) \cap \mathsf{bnd}(\mathcal{F}) = \emptyset$ are referred to as *non-conflicting* BESs.

As usual, we consider only equation systems $\mathcal{E}$ in which every proposition variable occurs at the left-hand side of at most one equation of $\mathcal{E}$. We define an ordering $\trianglelefteq$ on bound variables of

an equation system $\mathcal{E}$, where $X \trianglelefteq X'$ indicates that the equation for X precedes the equation for X'.

Proposition formulae are interpreted in a context of an *environment* $\eta{:}\mathcal{X} \to \mathbb{B}$. For an arbitrary environment η, we write $\eta[X := b]$ for the environment η in which the proposition variable X has Boolean value b.

Finding a solution of a BES amounts to finding an assignment of true or false to each variable X_i such that all equations are satisfied. Furthermore if $\sigma_i = \mu$, then the assignment to X_i is as strong as possible, and if $\sigma_i = \nu$ it is as weak as possible, where the leftmost equation takes priority over equations that follow. The concept of a solution is formalised below.

Definition 3.2.2. Let $\eta{:}\mathcal{X} \to \mathbb{B}$ be an environment. The *interpretation* $[\![f]\!]\eta$ maps a proposition formula f to true or false:

$$[\![c]\!]\eta \triangleq c \qquad [\![X]\!]\eta \triangleq \eta(X)$$
$$[\![f \vee g]\!]\eta \triangleq [\![f]\!]\eta \vee [\![g]\!]\eta \qquad [\![f \wedge g]\!]\eta \triangleq [\![f]\!]\eta \wedge [\![g]\!]\eta$$

Let η be an environment. Let $b_\mu = \mathsf{false}$ and $b_\nu = \mathsf{true}$. The *solution* of a BES, given η, is inductively defined as follows:

$$\begin{aligned} [\![\epsilon]\!]\eta &= \eta \\ [\![(\sigma X = f)\mathcal{E}]\!]\eta &= [\![\mathcal{E}]\!]\eta[X := f([\![\mathcal{E}]\!]\eta[X := b_\sigma])] \end{aligned}$$

We also write $\eta(X)$ to denote the interpretation of X in environment η. In the sequel, when we refer to solving a BES we mean computing the solution of the BES.

We introduce the following terminology.

Definition 3.2.3. Let $\mathcal{E}$ be an equation system. Then

- $\mathcal{E}$ is *closed* whenever $\mathsf{occ}(\mathcal{E}) \subseteq \mathsf{bnd}(\mathcal{E})$;
- $\mathcal{E}$ is *solved* whenever $\mathsf{occ}(\mathcal{E}) = \emptyset$;

For closed BES $\mathcal{E}$, $[\![\mathcal{E}]\!]\eta = [\![\mathcal{E}]\!]\eta'$ for arbitrary environments η and η', hence we may omit the environment in this case. Also observe that according to the semantics $\wedge$ and $\vee$ are commutative and associative, hence we may write *e.g.* $\bigwedge_{i=0}^{j} f_i$ instead of $f_0 \wedge \ldots f_n$, for formulae f_i.

In the remainder we restrict ourselves to closed BESs. For a closed BES $\mathcal{E}$ we define the right hand side rhs of a propositional variable $X \in \mathsf{bnd}(\mathcal{E})$ as the *right hand side* of the defining equation of X in $\mathcal{E}$:

$$\mathsf{rhs}(X, (\sigma Y = f)\mathcal{E}) \triangleq \begin{cases} f & \text{if } X = Y \\ \mathsf{rhs}(X, \mathcal{E}) & \text{otherwise} \end{cases}$$

Definition 3.2.4. A BES $\mathcal{E}$ is in simple form (SF) if every equation in $\mathcal{E}$ is of the form $\sigma X = f$, $\sigma X = \bigwedge_{i=0}^{n} f_i$ or $\sigma X = \bigvee_{i=0}^{n} f_i$, where $n > 0$, and f is either a propositional variable, or one of the Boolean constants true or false.

That is, a BES is in simple form if every right hand side is either a single variable or Boolean constant, or it is a conjunction or a disjunction over propositional variables or Boolean constants. Conjunctions and disjunctions may not appear mixed in a single right hand side. Note that every BES can be transformed into simple form in polynomial time in such a way that the variables in the original BES are preserved, and variables that occur in both BESs have the same solution, see [Mad97]. An equation can, for example, be transformed to simple form as follows. Given an equation $\sigma X = \bigwedge_{i=0}^{k} f_i$, and some f_j is disjunctive, replace this single equation by two equations $(\sigma X = \bigwedge_{i=0}^{j-1} \wedge X' \bigwedge_{i=j+1}^{k})(\sigma X' = f_j)$, where X' is fresh. The case for $\vee$ is analogous, and the transformation can be repeated until a BES in simple form is obtained.

We can also restrict a BES such that it does not contain Boolean constants. This is referred to as recursive form.

Definition 3.2.5. A BES $\mathcal{E}$ is in recursive form (RF) if the Boolean constants true and false do not occur in $\mathcal{E}$.

The transformation of a BES to a BES in RF can also be done in a solution preserving way, introducing auxiliary equations for Boolean constants true and false.

When we combine the notions of simple form and recursive form we obtain the concept of simple recursive form.

Definition 3.2.6. A BES $\mathcal{E}$ is in simple recursive form (SRF) if $\mathcal{E}$ is in simple form, and the Boolean constants true and false do not occur in $\mathcal{E}$.

The translation of a BES to SRF is simply the composition of the translations of a BES to SF and RF, and hence is also solution preserving.

Definition 3.2.7. A BES $\mathcal{E}$ is in conjunctive form if every equation in $\mathcal{E}$ is of the form $\sigma X = \bigwedge_{i=0}^{n} f_i$, with $n \geqslant 0$, and f_i a propositional variable or a Boolean constant.

That is, a BES in conjunctive form only contains conjuncts, single variables or Boolean constants as right hand sides. It has been shown [Mad97] that given a BES $\mathcal{E}$ and an environment η there is a BES $\mathcal{E}'$ in conjunctive form such that $\mathcal{E}$ and $\mathcal{E}'$ have the same solutions in η.

A similar notion is a BES in disjunctive form, *i.e.* a BES that only contains disjuncts, single variables or Boolean constants as right hand sides.

Definition 3.2.8. A BES $\mathcal{E}$ is in disjunctive form if every equation in $\mathcal{E}$ is of the form $\sigma X = \bigvee_{i=0}^{n} f_i$, with $n \geqslant 0$, and f_i a propositional variable or Boolean constant.

A derived notion of a closed equation system $\mathcal{E}$ is its *dependency graph* $\mathcal{G}_{\mathcal{E}}$, which is defined as a structure $\langle V, \rightarrow \rangle$, where:

- $V = \mathsf{bnd}(\mathcal{E})$;
- $X \rightarrow Y$ iff there is some equation $\sigma X = f$ in $\mathcal{E}$ with $Y \in \mathsf{occ}(f)$;

We introduce the notion of rank of an equation, and some derived notions. These notions are an indication of the complexity of the BES, as well as a measure that occurs in the computational complexity of some of the algorithms for solving BESs.

Definition 3.2.9. Let $\mathcal{E}$ be an arbitrary equation system. The *rank* of some $X \in \mathsf{bnd}(\mathcal{E})$, denoted $\mathsf{rank}(X)$, is defined as $\mathsf{rank}(X) = \mathsf{rank}_{\nu,X}(\mathcal{E})$, where $\mathsf{rank}_{\nu,X}(\mathcal{E})$ is defined inductively as follows:

$$\mathsf{rank}_{\sigma,X}(\epsilon) = 0$$

$$\mathsf{rank}_{\sigma,X}((\sigma' Y = f)\mathcal{E}) = \begin{cases} 0 & \text{if } \sigma = \sigma' \text{ and } X = Y \\ \mathsf{rank}_{\sigma,X}(\mathcal{E}) & \text{if } \sigma = \sigma' \text{ and } X \neq Y \\ 1 + \mathsf{rank}_{\sigma',X}((\sigma' Y = f)\mathcal{E}) & \text{if } \sigma \neq \sigma' \end{cases}$$

Observe that $\mathsf{rank}(X)$ is odd iff X is defined in a least fixpoint equation.

The *alternation hierarchy* of an equation system can be thought of as the number of syntactic alternations of fixpoint signs occurring in the equation system. The alternation hierarchy $\mathsf{ah}(\mathcal{E})$ of an equation system $\mathcal{E}$ can be defined as the difference between the largest and the smallest rank occurring in $\mathcal{E}$, formally $\mathsf{ah}(\mathcal{E}) = \max\{\mathsf{rank}(X) \mid X \in \mathsf{bnd}(\mathcal{E})\} - \min\{\mathsf{rank}(X) \mid X \in \mathsf{bnd}(\mathcal{E})\}$.

Given an equation $(\sigma X = f)$ in SF, the function $\mathsf{op}(X)$ returns whether f is conjunctive ($\wedge$), disjunctive ($\vee$) or neither ($\perp$);

An alternative characterisation of the solution of a particular proposition variable X in an equation system $\mathcal{E}$ in SRF is obtained through the use of the dependency graph $\mathcal{G}_{\mathcal{E}}$. We first define the notion of a ν-dominated lasso.

Definition 3.2.10. Let $\mathcal{E}$ be a closed equation system, and let $\mathcal{G}_\mathcal{E}$ be its dependency graph. A *lasso* through $\mathcal{G}_\mathcal{E}$, starting in a node X, is a finite path $\langle X_0, X_1, \ldots, X_n \rangle$, satisfying $X_0 = X$, $X_n = X_j$ for some $j \leq n$, and for each $1 < i \leq n$, $X_{i-1} \rightarrow X_i$. A lasso is said to be *ν-dominated* if $\min\{\mathsf{rank}(X_i) \mid j \leq i \leq n\}$ is even; otherwise, it is *μ-dominated*.

The following lemma is loosely based on lemmata taken from Keinänen (see lemmata 40 and 41 in [Kei06]).

Lemma 3.2.11. Let $\mathcal{E}$ be a closed equation system in SRF, and let $\mathcal{G}_\mathcal{E}$ be its dependency graph. Let $X \in \mathsf{bnd}(\mathcal{E})$. Then:

1. If $\mathcal{E}$ is *disjunctive*, then $[\![\mathcal{E}]\!](X) = \mathsf{true}$ iff some lasso starting in X in $\mathcal{G}_\mathcal{E}$ is ν-dominated;

2. If $\mathcal{E}$ is *conjunctive*, then $[\![\mathcal{E}]\!](X) = \mathsf{false}$ iff some lasso starting in X in $\mathcal{G}_\mathcal{E}$ is μ-dominated;

Proof We only consider the first statement; the proof of the second statement is analogous. Observe that when the proposition variable on the cycle of the lasso has an even lowest rank, it is a greatest fixpoint equation $\nu X' = f$, with $X' \trianglelefteq Y$ for all other equations $\sigma Y = g$ that are on the cycle. This follows from the fact that these have higher ranks. *Gauß elimination* [Mad97] allows one to substitute g for Y in the equation for X', yielding $\nu X' = f[Y{:=}g]$. Since, ultimately, X' depends on X' again, this effectively enables one to rewrite $\nu X' = f$ to $\nu X' = f' \vee X'$. The solution to $\nu X' = f' \vee X'$ is easily seen to be $X' = \mathsf{true}$. Since all equations on the lasso are disjunctive, this solution ultimately propagates through the entire lasso, leading to $X = \mathsf{true}$.

Conversely, observe that there is an equation system $\mathcal{E}'$ consisting entirely of equations of the form $\sigma X' = X''$ (follows from Corollary 3.37 in [Mad97]), with the additional property that $[\![\mathcal{E}]\!] = [\![\mathcal{E}']\!]$. In $\mathcal{E}'$, the answer to X can only be true if it depends at some point on some $\nu X' = X''$, where ultimately, X'' again depends on X', leading to a cycle in the dependency graph with even lowest rank. □

3.3 Relation between BES and games

In this section we investigate how Boolean Equation Systems and various kinds of graph games are related. We mainly focus our attention to the equivalence between BESs and parity games, but we first investigate the similar notion of graph games.

3.3.1 Equivalence of Boolean Equation Systems and Graph games

Mader [Mad97] showed the equivalence of Boolean Equation Systems and a variation of game graphs equivalent to Stirling games and parity games that we refer to as Mader games. The translation by Mader [Mad97] contains some small mistakes leading to incorrect results, hence we present a corrected version of the translation here for the sake of completeness. This translation also is the basis for the translation of BESs to parity games by Keinänen [Kei06].

A Mader game consists of a set of vertices $\{1, \ldots, n\}$. Each vertex gets two labels, one from $\{I, II\}$ and one from $\{\mu, \nu\}$. Furthermore, for each vertex i the graph contains at least one edge $i \rightarrow j$. A play π is an infinite sequence of moves starting with a token at some vertex i, such that player I has to move if the token is on a vertex labelled I, similarly if the token is on a vertex labelled II player II has to move.

Definition 3.3.1. [Mad97, Section 8.2] Consider the set of vertices $Inf(\pi)$, denoting the set of vertices of the Mader game, occurring infinitely often on play π. Let $w = min(Inf(\pi))$ be the least vertex that occurs infinitely often. Player I wins the play if vertex w is labelled μ, otherwise player II wins the play.

From Boolean Equation Systems to Mader games

Given a closed Boolean Equation System $\mathcal{E}$ in SRF, the corresponding game graph $\mathcal{G}^g_\mathcal{E}$ can be defined as follows. Let the game graph $\mathcal{G}^g_\mathcal{E}$ be the dependency graph of the $\mathcal{E}$, where each vertex is decorated with two labels:

- For every equation $\sigma X_i = f$ in $\mathcal{E}$, vertex i in $\mathcal{G}^g_\mathcal{E}$ is labelled with σ;
- if $\mathsf{op}(X_i) = \wedge$ then vertex i is labelled with I
- all other vertices are labelled with II.

As is shown by the following theorem, the translation preserves the solution in such a way that player II wins the game if and only if the BES has solution true.

Theorem 3.3.2.[Mad97] Player II has a winning strategy for the game on $\mathcal{G}^g_\mathcal{E}$ with initial vertex i iff $[\![\mathcal{E}]\!](X_i) = \mathsf{true}$.

From Mader games to Boolean Equation Systems

Given game graph $\mathcal{G}$, the corresponding Boolean Equation System $\mathcal{E}_\mathcal{G}$ in SRF can be obtained as follows.

- If vertex i of $\mathcal{G}$ is labelled σ, then there is an equation $\sigma X_i = f_i$ in $\mathcal{E}_\mathcal{G}$;
- if vertex i has label I and $S_i = \{j \mid i \to j \text{ is an edge is } \mathcal{G}\}$, then $\sigma X_i = \bigwedge_{j \in S_i} X_j$ is an equation of $\mathcal{E}_\mathcal{G}$;
- if vertex i has label II and $S_i = \{j \mid i \to j \text{ is an edge is } \mathcal{G}\}$, then $\sigma X_i = \bigvee_{j \in S_i} X_j$ is an equation of $\mathcal{E}_\mathcal{G}$;
- if $i < j$, then $X_i \lhd X_j$ in $\mathcal{E}_\mathcal{G}$.

Given game graph $\mathcal{G}$, the corresponding Boolean Equation System $\mathcal{E}_\mathcal{G}$ in SRF can be obtained as follows.

Theorem 3.3.3.[Mad97] Player II has a winning strategy for the game on $\mathcal{G}^g_\mathcal{E}$ with initial vertex i iff $[\![\mathcal{E}]\!](X_i) = \mathsf{true}$.

Note that in the original definition by Mader [Mad97] $\bigwedge$ and $\bigvee$ are exchanged in the translation from graph games to BESs, leading to an incorrect transformation as is shown by the following counterexample to Theorem 3.3.3.

Example 3.3.4. Consider the following BES:

$$\begin{aligned} \mu X &= Y \vee X \\ \nu Y &= X \wedge Y \end{aligned}$$

The Mader game constructed from this is the game with vertices X, Y, where X is labelled $\{\mu, II\}$, and Y is labelled $\{\nu, I\}$, furthermore there are edges $X \to X$, $X \to Y$, $Y \to X$ and $Y \to Y$.

When we apply the reverse translation (according to the original definition by Mader) we obtain the following BES:

$$\begin{aligned} \mu X &= Y \wedge X \\ \nu Y &= X \vee Y \end{aligned}$$

As we see, the $\wedge$ and $\vee$ symbols have been exchanged, and the solutions of both BESs differ. (As the first BES has solution $X = Y = \mathsf{false}$, whereas the second BES has solution $X = \mathsf{false}$, $Y = \mathsf{true}$.

3.3.2 Equivalence of Boolean Equation Systems and parity games

Given a closed BES $\mathcal{E}$ in SRF, we can find a parity game $\Gamma_{\mathcal{E}}$ such that $[\![\mathcal{E}]\!](X)$ is true for some variable X if and only if player *Even* has a winning strategy on $\Gamma_{\mathcal{E}}$ from vertex X [Kei06].

From Boolean Equation Systems to parity games

We translate a BES $\mathcal{E}$ to a parity game $\Gamma = (V, E, p, (V_{Even}, V_{Odd}))$ as follows. As game graph we take the dependency graph $\mathcal{G}_{\mathcal{E}}$ of $\mathcal{E}$. Furthermore priorities are assigned according to the ranks in the BES, i.e. $p(X) = \mathsf{rank}(X)$. Furthermore, all $X \in \mathsf{bnd}(\mathcal{E})$ with $\mathsf{op}(X) = \wedge$ are assigned to V_{Odd}, the other vertices to V_{Even}.

From parity games to Boolean Equation Systems in SRF

Analogously, we translate a parity game to a BES as follows. For each vertex $v \in V$, with edges $(v, w_1), \ldots (v, w_{n_v})$, we create the following equation. If v in V_{Odd}, we create $\sigma X_v = \bigvee_{i=0}^{n_v} X_i$, for vertices v in V_{Even}, equation $\sigma X_v = \bigwedge_{i=0}^{n_v} X_i$ is introduced. In both bases $\sigma = \mu$ if $p(v)$ is odd, and $\sigma = \nu$ otherwise. Furthermore, if $p(v) < p(u)$ for two vertices v and u in the parity game, it holds that $X_v \lhd X_u$ in the BES.

Theorem 3.3.5. [Kei06] Player *Even* has a winning strategy for $\mathcal{G}_{\mathcal{E}}$ from vertex X_i if and only if $([\![\mathcal{E}]\!]\eta)(X_i) = \mathsf{true}$.

3.3.3 Simplification on Boolean Equation Systems

Mader has shown a large number of equivalences on Boolean equation systems [Mad97]. We investigate how the simplifications for parity games coincide with the knowledge about BESs. Some of the simplifications for parity games give rise to new results in the framework of BESs.

We give an overview of the BES related theory that corresponds to the parity game theory from Section 3.1.1. In the cases where new simplifications are introduced we also give proofs directly on BESs.

3.3.4 Reducing the sizes of formulae

In the literature, *e.g.* [Mad97] techniques are described that reduce the size of formulae in the right hand sides of an equation.

Local resolution Using the semantics of an equation in a BES, self-references can be removed from the right hand side of an equation. This technique is referred as local resolution. Some local resolution steps in BES coincide with the elimination of self-loops in parity games. Consider the following equivalences:

$$\begin{aligned} \mu X = X \vee Y &\equiv \mu X = Y \\ \nu X = X \wedge Y &\equiv \nu X = Y \\ \mu X = X \wedge Y &\equiv \mu X = \mathsf{false} \\ \nu Y = X \vee Y &\equiv \nu Y = \mathsf{true} \end{aligned}$$

Proofs that these equivalences hold can be found in [Mad97]. The first two rules coincide with the cases in which a vertex in a parity game has a self-loop, and the evenness of the priority and the player differ, hence a self-loop can be removed. The other two rules coincide with the cases in which a trivial dominion is found, and hence part of the game is solved.

Note that in general right hand sides in BESs can be simplified using equivalence results on Boolean formulae known from the literature, and hence such simplifications are not restricted to the four that we have mentioned.

It is evident that the simplifications in this section reduce the cumulative size of the right hand sides in a BES, and hence also the size of the BES.

Rank reducing simplifications

In the discussion of parity games we have seen some simplifications that reduce the number of priorities in the game. Investigating similar operations in the setting of BESs gives rise to optimisations in BES that have not occurred in the literature yet.

Observe that the priority compaction technique on parity games does not have a natural corresponding manipulation on Boolean Equation Systems, as we choose the rank of an equation as priority in the corresponding parity game.

Irrelevance of fixpoint operator The priority propagation technique leads to a new equivalence for Boolean Equation Systems. Observe that in priority propagation, the priority of a node v in the parity game changes if either all of its predecessors or all of its successors have a priority that is lower than the priority of v. When we also take into account the reduction of a BES in SRF to a parity game, observe that we use the rank of an equation as its priority. This gives rise to the following observation. If, in the parity game corresponding to a BES $\mathcal{E}$ there are priorities that change from even to odd, or from odd to even, then there are equations in $\mathcal{E}$ of which the fixpoint symbol may be changed, preserving the solution of $\mathcal{E}$. This follows immediately from translating $\mathcal{E}$ to parity game, applying priority propagation on the game, and translating the resulting game back to BES. Next we will show the correctness of this technique directly on BESs.

In general, if the defining equations of all variables that occur in the right hand side of an equation $\sigma X = f$ occur before $\sigma X = f$ in the BES, then we may replace $\sigma X = f$ with $\sigma' X = f$, where σ' is an arbitrary fixpoint symbol. Similarly, if X only occurs in equations that occur after $\sigma X = f$ in the BES we may also choose an arbitrary fixpoint.

Lemma 3.3.6. Let η be an arbitrary environment, and $\mathcal{E}_1, \mathcal{E}_2$ be arbitrary non-conflicting BESs. Then

$$[\![\mathcal{E}_1(\sigma X = f)\mathcal{E}_2]\!]\eta = [\![(\sigma' X = f)\mathcal{E}_1\mathcal{E}_2]\!]\eta$$

provided that $X \notin \mathsf{occ}(\mathcal{E}_1) \cup \mathsf{occ}(\mathcal{E}_2)$ and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

Proof We prove this by induction on the length of $\mathcal{E}_1$.

Case $\mathcal{E}_1 = \epsilon$) We show that

$$[\![(\sigma X = f)\mathcal{E}_2]\!]\eta = [\![(\sigma' X = f)\mathcal{E}_2]\!]\eta$$

provided that $X \notin \mathsf{occ}(\mathcal{E}_2)$, and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

$$\begin{array}{ll} & [\![(\sigma X = f)\mathcal{E}_2]\!]\eta \\ = & \{\text{Semantics}\} \\ & [\![\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_2]\!]\eta[X := b_\sigma])] \\ = & \{X \notin \mathsf{occ}(\mathcal{E}_\epsilon), \text{ and } X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'\} \\ & [\![\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_2]\!]\eta[X := b_{\sigma'}])] \\ = & \{\text{Semantics}\} \\ & [\![(\sigma' X = f)\mathcal{E}_2]\!]\eta \end{array}$$

Case $\mathcal{E}_1 = (\sigma_Y Y = g)\mathcal{E}_1'$) We use as induction hypothesis that

$$[\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta = [\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta$$

for arbitrary η, provided that $X \notin \mathsf{occ}(\mathcal{E}_1) \cup \mathsf{occ}(\mathcal{E}_2)$ and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

$$
\begin{aligned}
& [\![(\sigma_Y Y = g)\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta \\
= \quad & \{\text{Semantics}\} \\
& [\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])] \\
= \quad & \{\text{Introduce } \eta_Y \stackrel{\Delta}{=} \eta[Y := g([\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])]\} \\
& [\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta_Y \\
= \quad & \{\text{Induction hypothesis}\} \\
& [\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y \\
= \quad & \{\text{Semantics}\} \\
& [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := b_{\sigma'}])] \\
= \quad & \{\text{Definition of } \eta_Y\} \\
& [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := b_{\sigma'}])] \\
= \quad & \{\text{Induction hypothesis}\} \\
& [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := b_{\sigma'}])] \\
= \quad & \{\text{Semantics, use that } X \notin \mathsf{occ}(g) \cup \mathsf{occ}(\mathcal{E}_1') \cup \mathsf{occ}(\mathcal{E}_2)\} \\
& [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := b_{\sigma'}])] \\
= \quad & \{\text{Semantics}\} \\
& [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := b_{\sigma'}])] \\
= \quad & \{\text{Definition of } \eta_Y\} \\
& [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'(\sigma X = f)\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := b_{\sigma'}])] \\
= \quad & \{\text{Induction hypothesis}\} \\
& [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := b_{\sigma'}])] \\
= \quad & \{\text{Semantics, use that } X \notin \mathsf{occ}(g) \cup \mathsf{occ}(\mathcal{E}_1') \cup \mathsf{occ}(\mathcal{E}_2)\} \\
& [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][X := b_{\sigma'}])] \\
= \quad & \{\text{Semantics}\} \\
& [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := b_{\sigma'}])] \\
= \quad & \{\text{Semantics}\} \\
& [\![(\sigma' X = f)(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta
\end{aligned}
$$

□

This lemma can be generalised as follows.

Theorem 3.3.7. Let η be an arbitrary environment, and $\mathcal{E}_0, \mathcal{E}_1$ and $\mathcal{E}_2$ be arbitrary non-conflicting BESs. Then

$$[\![\mathcal{E}_0\mathcal{E}_1(\sigma X = f)\mathcal{E}_2]\!]\eta = [\![\mathcal{E}_0(\sigma' X = f)\mathcal{E}_1\mathcal{E}_2]\!]\eta$$

provided that $X \notin \mathsf{occ}(\mathcal{E}_1) \cup \mathsf{occ}(\mathcal{E}_2)$ and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

Proof Follows immediately from Lemma 3.3.6 and Lemma 3.14 from Mader [Mad97]. □

Similarly, we show that if all variables that occur in an equation are defined before that equation, we may reposition it and choose an arbitrary fixpoint symbol.

Lemma 3.3.8. Let η be an arbitrary environment, and $\mathcal{E}_1$ and $\mathcal{E}_2$ arbitrary non-conflicting BESs. Then

$$[\![(\sigma X = f)\mathcal{E}_1\mathcal{E}_2]\!]\eta = [\![\mathcal{E}_1(\sigma' X = f)\mathcal{E}_2]\!]\eta$$

provided that $\mathsf{occ}(f) \cap \mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2) = \emptyset$, and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

Proof We prove this by induction to the length of $\mathcal{E}_1$.
Case $\mathcal{E}_1 = \varepsilon$). We show that

$$[\![(\sigma X = f)\mathcal{E}_2]\!]\eta = [\![(\sigma' X = f)\mathcal{E}_2]\!]\eta$$

provided that $\mathsf{occ}(f) \cap \mathsf{bnd}(\mathcal{E}_2) = \emptyset$ and $X \in \mathsf{occ}(f) \rightarrow \sigma = \sigma'$.

$$\begin{array}{ll} & [\![(\sigma X = f)\mathcal{E}_2]\!]\eta \\ = & \{\text{Semantics}\} \\ & [\![\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_2]\!]\eta[X := b_\sigma])] \\ = & \{\forall Z \in \mathsf{occ}(f) : [\![\mathcal{E}_2]\!]\eta[X := b_\sigma](Z) = [\![\mathcal{E}_2]\!]\eta[X := b_{\sigma'}](Z)\} \\ & [\![\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_2]\!]\eta[X := b_{\sigma'}])] \\ = & \{\text{Semantics}\} \\ & [\![(\sigma' X = f)\mathcal{E}_2]\!]\eta \end{array}$$

Case $\mathcal{E}_1 = (\sigma_Y Y = g)\mathcal{E}_1'$). As induction hypothesis we use

$$[\![(\sigma X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta = [\![\mathcal{E}_1'(\sigma' X = f)\mathcal{E}_2]\!]\eta$$

provided that $\mathsf{occ}(f) \cap \mathsf{bnd}(\mathcal{E}_1'\mathcal{E}_2) = \emptyset$, and $X \in \mathsf{occ}(f) \implies \sigma = \sigma'$.

$$\begin{array}{ll} & [\![(\sigma X = f)(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta \\ = & \{\text{Semantics}\} \\ & [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := b_\sigma])] \\ = & \{\text{Use: } \forall Z \in \mathsf{occ}(f) : \eta(Z) = [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := b_\sigma](Z)\} \\ & [\![(\sigma_Y Y = g)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f(\eta)] \\ = & \{\text{Semantics}\} \\ & [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f(\eta)][Y := g([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f(\eta)][Y := b_{\sigma_Y}])] \\ = & \{\text{Use: } \forall Z \in \mathsf{occ}(f) : \eta(Z) = [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}](Z)\} \\ & [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f(\eta)][Y := g([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f([\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])][Y := b_{\sigma_Y}] \\ = & \{\text{Semantics}\} \\ & [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta[X := f(\eta)][Y := g([\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])] \\ = & \{\text{Introduce } \eta_Y \stackrel{\Delta}{=} \eta[Y := g([\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])]\} \\ & [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := f(\eta)] \\ = & \{\text{Use: } \forall Z \in \mathsf{occ}(f) : \eta(Z) = \eta_Y(Z)\} \\ & [\![\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y[X := f(\eta_Y)] \\ = & \{X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'\} \\ & [\![(\sigma' X = f)\mathcal{E}_1'\mathcal{E}_2]\!]\eta_Y \\ = & \{\text{Induction hypothesis (twice)}\} \\ & [\![\mathcal{E}_1'(\sigma' X = f)\mathcal{E}_2]\!]\eta[Y := g([\![\mathcal{E}_1'(\sigma' X = f)\mathcal{E}_2]\!]\eta[Y := b_{\sigma_Y}])] \\ = & \{\text{Semantics}\} \\ & [\![(\sigma_Y Y = g)\mathcal{E}_1'(\sigma' X = f)\mathcal{E}_2]\!]\eta \end{array}$$

□

Lemma 3.3.8 generalises to the following theorem.

Theorem 3.3.9. Let η be an arbitrary environment, and $\mathcal{E}_0, \mathcal{E}_1$ and $\mathcal{E}_2$ arbitrary non-conflicting BESs. Then

$$[\![\mathcal{E}_0(\sigma X = f)\mathcal{E}_1\mathcal{E}_2]\!]\eta = [\![\mathcal{E}_0\mathcal{E}_1(\sigma' X = f)\mathcal{E}_2]\!]\eta$$

provided that $\mathsf{occ}(f) \cap \mathsf{bnd}(\mathcal{E}_1\mathcal{E}_2) = \emptyset$, and $X \in \mathsf{occ}(f) \Rightarrow \sigma = \sigma'$.

Proof This theorem follows immediately from Lemma 3.3.8 and Lemma 3.14 from Mader [Mad97]. □

Algorithms for special cases

Similar to the parity game environment, in the setting of BES some efficient algorithms for subsets of BES are known. These could in the BES setting be used to speed up computation, provided that the special cases can be detected efficiently.

Alternation free BES The single parity parity games, for which we know that linear time algorithms exist, correspond to special cases of alternation free BESs, *i.e.* BESs $\mathcal{E}$ with $\mathsf{ah}(\mathcal{E}) = 1$. To be more precise, a single parity parity game with odd parity corresponds to an alternation free BES in SRF in which each equation has μ as fixpoint symbol. Analogously, every single parity parity game with even parity corresponds to an alternation free BES in SRF with only ν as fixpoint symbol. Observe that these classes of Boolean Equation Systems are always false or true respectively.

It is known that alternation free BESs can be solved in time linear in the number of equations and the cumulative size of the right hand sides in the BES [Mat03].

Conjunctive/Disjunctive BES Single player games, which can also be solved in linear time, correspond to the classes of conjunctive and disjunctive Boolean Equation Systems. A parity game in which all vertices with more than one outgoing edge belong to player *Even* corresponds to a disjunctive BES, a parity game in which all vertices with more than one outgoing edge are owned by *Odd* corresponds to a conjunctive BES. For this class of Boolean Equation Systems an algorithm running in $O(|\mathcal{E}| \cdot \mathsf{ah}(\mathcal{E}))$ is known [GK05]. Note that technically, the equation $\sigma X = X$ can be translated to a vertex v in a parity game with either $v \in V_{Even}$ or $v \in V_{Odd}$.

SCC decomposition

As observed by Keinänen [Kei06], SCC decomposition can also be used to speed up BES solving.

Given a BES $\mathcal{E}$ in simple form, let $\mathcal{Y} \subseteq \mathcal{X}$ be a set of propositional variables defined in $\mathcal{E}$, furthermore $\oplus \in \{\wedge, \vee\}$, where $\overline{\wedge} = \vee$ and $\overline{\vee} = \wedge$. We denote with $\mathcal{X}_\oplus$ the variables with a formula containing $\oplus$ as right hand side. Note that we consider a formula without Boolean connectives to be part of both $\mathcal{X}_\wedge$ and $\mathcal{X}_\vee$. $\mathcal{X}_\oplus$ is formally defined as $\mathcal{X}_\oplus = \{X \mid X \in \mathsf{bnd}(\mathcal{E}) \wedge (\mathsf{op}(X) = \oplus \vee \mathsf{op}(X) = \bot)\}$. We define the attractor sets inductively as follows:

$$
\begin{aligned}
\mathit{Attractor}^0_\oplus(\mathcal{Y}) &= \mathcal{Y} \\
\mathit{Attractor}^{k+1}_\oplus(\mathcal{Y}) &= \mathit{Attractor}^k_\oplus(\mathcal{Y}) \\
&\cup \ (\mathcal{X}_\oplus \cap \{Y \mid \mathsf{occ}(\mathsf{rhs}(Y, \mathcal{E})) \cap \mathit{Attractor}^k_\oplus(\mathcal{Y}) \neq \emptyset\}) \\
&\cup \ (\mathcal{X}_{\overline{\oplus}} \cap \{Y \mid \mathsf{occ}(\mathsf{rhs}(Y, \mathcal{E})) \subseteq \mathit{Attractor}^k_\oplus(\mathcal{Y})\}) \\
\mathit{Attractor}_\oplus(\mathcal{Y}) &= \bigcup_{k \in \mathbb{N}} \mathit{Attractor}^k_\oplus(\mathcal{Y})
\end{aligned}
$$

Intuitively the attractor sets are the sets of variables that can be determined to have the same solution as the variables in $\mathcal{Y}$. More specifically, if all variables in $\mathcal{Y}$ are known to have solution false, $\mathit{Attractor}_\wedge(\mathcal{Y})$ is the set of variables that can then also be determined to be false. Similarly if all variables in $\mathcal{Y}$ are true all variables in $\mathit{Attractor}_\vee(\mathcal{Y})$ are also true. Other combinations of operators and sets of variables known to be true or false are less meaningful, hence we do not consider them.

We can then optimise solving Boolean Equation Systems as follows. Compute the strongly connected components of the dependency graph underlying the BES. Solve the equations per terminating SCC using a BES solving algorithm. These solutions can be used as solutions in the rest of the BES.

Next the attractors for $\{\vee, \wedge\}$ can be computed with respect to the sets of variables that have become true and false. These attractors get the corresponding solutions and can also be removed from the BES. The result remains a BES, but the SCCs may be altered, hence we decompose the

altered SCCs into SCCs again, resulting in a new decomposition. This process is repeated until the entire BES has been solved.

3.4 Bisimulation reduction

A well-known strategy for combating the state space explosion problem is reducing the state space prior to model checking. Minimising the state space is a well-known strategy that is used in this area. In practice the time required for performing bisimulation reduction followed by executing the model checking algorithm on the reduced state space often exceeds the running time required for model checking the original state space.

Though they seem slightly out of place in the setting of BESs, similar techniques can be applied. The work done in [KW09] however shows that this work is relevant. In this setting it is also shown that for the BES setting it is in most cases beneficial to apply the reduction before solving the BES. We will briefly summarise the relevant results.

Definition 3.4.1. Let $\mathcal{E}, \mathcal{E}'$ be closed equation systems in SRF. A relation $R \subseteq \mathsf{bnd}(\mathcal{E}) \times \mathsf{bnd}(\mathcal{E}')$ is said to be a *bisimulation* if, whenever XRY, then:

- $\mathsf{rank}(X) = \mathsf{rank}(Y)$ and $\mathsf{op}(X) = \mathsf{op}(Y)$;
- for all $U \in \mathsf{occ}(X)$, there is a $V \in \mathsf{occ}(Y)$, such that URV;
- for all $V \in \mathsf{occ}(Y)$, there is a $U \in \mathsf{occ}(X)$, such that URV;

We say equations for X and Y are *bisimilar*, denoted $X \sim Y$, if there exists a bisimulation relation R such that XRY; we say $\mathcal{E}$ and $\mathcal{E}'$ are bisimilar, denoted $\mathcal{E} \sim \mathcal{E}'$, if their first equations are bisimilar.

The following lemma shows that variables that are bisimulation equivalent have the same solution.

Lemma 3.4.2. Let $\mathcal{E}$ be an arbitrary closed BES in SRF, and X and Y variables in $\mathsf{bnd}(\mathcal{E})$, then

$$X \sim Y \Rightarrow [\![E]\!](X) = [\![E]\!](Y)$$

For every BES $\mathcal{E}$ in SRF, there is corresponding BES $\mathcal{E}_{/\sim}$, such that $\forall X \in \mathsf{bnd}(\mathcal{E}), Y \in \mathsf{bnd}(\mathcal{E}_{/\sim}) : X \sim Y \implies [\mathcal{E}](X) = [\mathcal{E}_{/\sim}](Y)$. In other words, $\sim$ can be quotiented, and it preserves solution equivalence.

In addition a variation of strong bisimulation which provides a coarser partitioning called oblivious bisimulation can be constructed. Oblivious bisimulation can be computed in the same running time as strong bisimulation but provides a coarser partitioning by sometimes identifying equations with different Boolean connectives in their right hand sides. Furthermore it has more pleasing theoretic properties when used for quotienting.

Reductions using strong bisimulation as well as oblivious bisimulation can be achieved in $O(m \log n)$ time using the Paige-Tarjan partition refinement algorithm [PT87].

3.5 Summary

In this section we have introduced the equivalence of parity games and BESs. Furthermore we have investigated optimisation techniques that exist in the parity game framework. We have investigate how similar optimisations recur in the framework of BESs. Where no similar techniques were available we have introduced new theory that accomplishes simplifications similar to the simplifications in parity games. We have also briefly introduced bisimulation reduction on BESs as this is a promising technique for improving the performance of BES solving.

Chapter 4

Overview of parity game algorithms

In Chapter 2 we described that we are interested in solving Boolean Equation Systems for model checking problems and other verification problems. Our experience shows that algorithms like Gauß elimination, and approximation (see *e.g.* [Mad97]) are not sufficiently efficient for the more complex model checking problems, *viz.* solving BESs with alternation. We therefore investigate the algorithms for solving parity games that are known in the literature. It follows from the previous chapter that these algorithms can also be applied to BESs in SRF, hence they might be an improvement for solving BESs in comparison to the known algorithms for solving BESs.

In this chapter we give a brief description of the parity game algorithms known in the literature [Lan05, FL09, McN93, Zie98, VJ00, SV00, Sch08, Jur00, JPZ06, SS98, Sch07]. Characteristic properties like worst case running time and space bounds are given. For all global algorithms examples are known where the algorithm achieves its worst case running time bound. For two of the algorithms this is a fairly recent result, see [Fri09].

We use the following conventions in the running time and memory bounds. We denote the number of vertices with n, the number of edges with e, and the number of priorities with d. All of the algorithms that are described here are implemented in the PGSolver toolset [FL09].

The parity game algorithms can be divided into several categories. First there are some algorithms based on fixpoint computation in Section 4.1. Furthermore there are some encodings of the problem into satisfiability described in Section 4.2. Section 4.3 presents some recursive algorithms that stem directly from constructive proofs of determinacy in the literature. Section 4.4 concludes with one algorithm for local model checking; the others are all global algorithms. Subsequently a summary of the algorithms with an hypothesis of the performance of the algorithms in practice is given in Section 4.5. This hypothesis is tested by the experiments in Chapter 5.

4.1 Fixed point algorithms

The algorithms with the most favourable theoretic running time bounds are the algorithms based on fixpoint computations. They assign a measure to each vertex, and transform that measure until a fixpoint is reached. From the resulting measure, winning sets and strategies can be computed.

4.1.1 Small progress measures algorithm [Jur00]

The small progress measures algorithm by Jurdziński [Jur00] attaches to each vertex a tuple of natural numbers. The values of these tuples are bound, and can be found iteratively starting from 0. When a fixpoint is reached, these tuples can be used to find a winning strategy for one of the players. This algorithm is referred to as `smallprog`. The small progress measures algorithm runs in $O(d \cdot e \cdot (\frac{n}{d})^{d/2})$ time and $O(d \cdot n \cdot \log n)$ space. Jurdziński also gives a class of parity games

on which this algorithm indeed has this worst case running time. This algorithm is presented in more detail in Chapter 6.

4.1.2 Strategy improvement algorithm [VJ00, SV00]

The strategy improvement algorithm by Vöge and Jurdziński [VJ00] selects an initial strategy for player *Even*. Then in each step a valuation of the current strategy is computed. Using this valuation the strategy for player *Even* is updated by choosing transitions that are locally optimal for player *Even*. This process is iterated until a fixpoint is reached. Subsequently winning sets and winning strategies are inferred from the final valuation. A description of an implementation of the algorithm was presented by Schmitz and Vöge [SV00]. We refer to this algorithm as `stratimprove`. The algorithm runs in $O(2^e \cdot n \cdot e)$ time and $O(n^2 + n \cdot \log d + e)$ space. It has been unknown for quite some time whether this algorithm was polynomial. However, Friedmann [Fri09] showed the existence of a class of examples where the algorithm indeed has a running time matching this bound, showing that the algorithm is exponential.

4.1.3 Optimal strategy improvement method [Sch08]

The strategy improvement algorithm by Vöge and Jurdziński suffers from the problem that an update that seems beneficial locally in one step, turns out to be the worst possible improvement when considering the game globally. The optimal strategy improvement method due to Schewe [Sch08] overcomes this problem, and guarantees in each improvement step that a globally optimal combination of local modifications is made. Again updates are made until a fixpoint is reached. We refer to this algorithm as `optstratimprov`. The algorithm runs in $O(e \cdot (\frac{n+d}{d})^d \cdot \log(\frac{n+d}{d}))$ time and $O(n^2)$ space (observe that $d < n$ and $e \leqslant n^2$, hence the space for strategy improvement could also be taken to be $O(n^2)$, and coincide with the space for the optimal strategy improvement algorithm). With respect to computational complexity, this algorithm has a history similar to the strategy improvement algorithm; it has been unknown whether the algorithm was polynomial for some time. Along with the examples obtaining the worst case running time for strategy improvement, Friedmann [Fri09] also showed that there are examples where the optimal variant performs according to its worst case running time bounds.

4.2 Satisfiability encodings

All algorithms in this section are encodings of the problem of solving parity games into satisfiability, *i.e.* the problem is encoded as a Boolean formula, for which a satisfying assignment needs to be found. As the problem is known to be in *NP* such an encoding must exist, but the efficiency of the encoding is not fixed.

4.2.1 Small progress measures encoding [Lan05]

The small progress measures algorithm induces a straightforward encoding of the parity game problem into satisfiability. A description of the encoding has been presented by Lange [Lan05] and is related to similar work in the BES setting by Keinänen [Kei06]. We refer to this algorithm as `viasat`. The algorithm runs in time $O(e \cdot d)$ plus the running time needed for the SAT solver. The space complexity is $O(e \cdot d)$ plus the space needed for the SAT solver.

4.2.2 Strategy improvement encoding [FL09]

Similar to the small progress measures algorithm, the strategy improvement algorithm attaches a measure to each vertex. Using fixpoint computation a value of the measure is computed from which the solution for the parity game can be inferred. This leads to a strategy improvement encoding that uses techniques similar to the encoding for small progress measures. This variant only occurs in the literature as an informal description by Friedmann and Lange [FL09]. We refer

to this algorithm as `stratimprsat`. The algorithm runs in time $O(n^2)$ and space $O(n^2)$ plus the time and space needed for the SAT solver.

4.2.3 Direct reduction [FL09]

The direct reduction to satisfiability formalises the existence of strategies, and requires the strategies to be winning by checking whether all cycles that can be reached by following the strategy of player *Player* are good for *Player*. No detailed description of the encoding is available, but some hints describing the algorithm are given by Friedmann and Lange [FL09]. We refer to this algorithm as `satsolve`. The algorithm runs in time $O(n^3)$ plus the running time needed for the SAT solver. The space complexity is $O(n^3)$ plus the space needed for the SAT solver.

4.3 Recursive algorithms

The algorithms in this section are inspired by inductive proofs of determinacy in the literature, and are characterised by their recursive nature.

4.3.1 Recursive algorithm [McN93, Zie98]

The determinacy proof for parity games as given by McNaughton [McN93] and Zielonka [Zie98] gives rise to a recursive algorithm for solving parity games. The game is decomposed into smaller subgames using induction on the number of priorities and the number of nodes in the game. The base cases are single player and single parity games, where a strategy can be obtained in a straightforward manner. In the other cases strategies are obtained out of the strategies of smaller subgames. This algorithm is referred to as `recursive`. The algorithm runs in $O(e \cdot n^d)$ time, and requires $O(e \cdot n)$ space.

4.3.2 Recursive preservation algorithm [FL09]

The recursive algorithm disregards already computed information at some points during the execution. The recursive preservation algorithm tries to preserve as much of this information as possible. This is an optimisation by Friedmann [FL09], but the full details remain unpublished to this date. We refer to this algorithm as `recpreserve`. The worst case running times and space of the recursive algorithm are not affected by the optimisation, hence this algorithm also runs in $O(e \cdot n^d)$ time, and requires $O(e \cdot n)$ space.

4.3.3 Dominion decomposition algorithm [JPZ06]

The dominion decomposition algorithm by Jurdziński, Paterson and Zwick [JPZ06] is similar to the recursive algorithm, and based on the same principles. Instead of immediately recursing it first tries to identify dominions (of size $\lceil \sqrt{n} \rceil$). If there is a dominion, the attractor set is constructed and, as the vertices in the attractor set are solved, they are removed from the game. The remaining subgame is solved recursively. If no dominion is found, the recursive algorithm is used. In successive steps the search for dominions is enabled again. We refer to this algorithm as `dominiondec`. The dominion decomposition algorithm runs in time $O(n^{O(\sqrt{n})})$ and space $O(e \cdot n)$. Observe that the running time of this algorithm is independent of the number of priorities in the game.

4.3.4 Big step algorithm

The big step algorithm due to Schewe [Sch07] is a refinement of the dominion decomposition algorithm that uses a restricted version of small progress measures algorithm finding small dominions. We refer to this algorithm with `bigstep`. The algorithm runs in time $O(e \cdot n^{\frac{d}{3}})$ and space $O((e + d \cdot \log n) \cdot n)$.

4.4 Local algorithms

Whereas the algorithms we have seen so far are designed to find winners for all vertices, *i.e.* provide global solutions to the model checking problem. The algorithm in this section is local.

4.4.1 Local model checking algorithm [SS98]

The local model checking algorithm by Stevens and Stirling [SS98] is based on depth-first search with backtracking, making sure all possible choices for the losing player have been considered. We refer to this algorithm as `modelchecker`. The literature does not give any sensible bounds for the running time and space requirements of this algorithm.

4.5 Summary

To summarise all running times we list all the algorithms given in this chapter with their running time and memory bounds in Table 4.1. The table also introduces abbreviations for the algorithms that will be used in the presentation of our experimental results in Chapter 5.

In the setting of model checking the modal μ-calculus, we expect d to be much smaller that n and e in general (in fact even smaller than $\sqrt{n}$), hence we expect the dominion decomposition algorithm to be the algorithm with worst performance. Furthermore, the small progress measures algorithm is expected to perform better than the optimal strategy improvement algorithm, which then again is expected to outperform the strategy improvement algorithm. Note that the expectancy of the strategy improvement algorithm to be outperformed by the optimal strategy improvement algorithm is also fed by the comparison between the two algorithms presented by Schewe [Sch08]. We also expect that the recursive and recursive preservation algorithms have performance similar to the optimal strategy improvement algorithm. The big step algorithm is expected to outperform all other algorithms in most cases. We do not make any prediction on the algorithms using satisfiability encoding as they are highly dependent on the performance of the SAT solver on the specific encoding, which we consider to be out of the scope of this thesis.

Table 4.1: Running time and memory bounds of parity game algorithms

Algorithm	**Running time**	**Space**
`viasat (vs)`	$O(e \cdot d)$ + time for SAT solver	$O(e \cdot d)$ + space for SAT solver
`stratimprsat (is)`	$O(n^2)$ + time for SAT solver	$O(n^2)$ + space for SAT solver
`satsolve (ss)`	$O(n^3)$ + time for SAT solver	$O(n^3)$ + space for SAT solver
`smallprog (sp)`	$O(d \cdot e \cdot (\frac{n}{d})^{d/2})$	$O(d \cdot n \cdot \log n)$
`stratimprove (si)`	$O(2^e \cdot n \cdot e)$	$O(n^2 + n \cdot \log d + e)$
`optstratimprove (os)`	$O(e \cdot (\frac{n+d}{d})^d \cdot \log(\frac{n+d}{d}))$	$O(n^2)$
`recursive (re)`	$O(e \cdot n^d)$	$O(e \cdot n)$
`recpreserve (rp)`	$O(e \cdot n^d)$	$O(e \cdot n)$
`dominiondec (dd)`	$O(n^{O(\sqrt{n})})$	$O(e \cdot n)$
`bigstep (bs)`	$O(e \cdot n^{\frac{d}{3}})$	$O((e + d \cdot \log n) \cdot n)$
`modelchecker (mc)`	unknown	unknown

Chapter 5

Experimental comparison of parity game algorithms

The large scala of parity game solving algorithms leads one to wonder about their performance in practical model checking problems. We do not know of any large scale comparison between parity game algorithms. It is however good to note that some experiments can be found in the literature [Sch08]. We have also described some optimisations for solving parity games. Of these optimisations no experiments showing their usefulness are available in the literature either.

Additionally, we show how much the algorithms suffer from an increasing number of priorities by varying the number of priorities in a single example. This shows how the dependency on the number of priorities of some algorithms manifests itself in practice.

We present an analyse a series of experiments that gives an insight in the practical aspects of parity game solvers.

Setup All experiments were run on a workstation consisting of 8 Dual Core [1] AMD Opteron(tm) processors running at 2.6 GHz, with 128GB of shared main memory, running a 64-bit Linux distribution using kernel version 2.6.24.

All parity games have been generated using the mCRL2 toolset[2] [GMWU07], by first producing a BES, and then converting the BES to a parity game using the prototype tool `paritygame` that employs the translation from Section 3.3.2. The parity games were subsequently solved with version 2.0 of the PGSolver toolset[3]. The solving process was terminated after 30 minutes, as for most cases there were algorithms that terminated well within that time limit.

For our experiments we consider model checking examples, *i.e.* given a model of a system and a property expressed as a modal formula, does the model satisfy the property. In addition, we treat the encoding of the branching bisimulation problem into PBES [CPPW07]. Both problems are described in Chapter 2.

Specifications The set of examples that we have used consists of a number of well known communication protocols, *viz.*, two variants of the Alternating Bit Protocol (ABP), the Concurrent Alternating Bit Protocol (CABP), the Sliding Window Protocol (SWP) with window sizes 2 and 3, the Bounded Retransmission Protocol (BRP), and the Onebit Protocol (OP). In these communication protocols the number of messages is a parameter, *i.e.* they consider a set of messages $M = \{d1, \ldots dn\}$, in the specifications we refer to the set of messages with D. With $|M|$ we refer to the number of messages.

Furthermore, a number of examples from the mCRL2 toolset have been considered, *viz.*, the physical layer service of the 1394 protocol (1394), a chat box (Chatbox), some variants of the

[1] None of our experiments employ multi-core features
[2] See `http://www.mcrl2.org`, revision 5832 (trunk)
[3] See `http://www.tcs.ifi.lmu.de/pgsolver/`

dining philosophers problem (Dining), a game called domineering (Domineering), a process with states $1, \ldots, 1000$, where each state i has transitions to all states $1, \ldots, , i+1$ (Goback), two variants of a leader election protocol (Leader), two variants of a system for lifting trucks (Lift), a controller for Movable Patient Support Unit (MPSU), a simple Producer-Consumer protocol (PC), the game Snake (Snake), Milner's well known scheduler (Scheduler), and some variants of Petterson's mutual exclusion protocol, modelled as a pair of trains entering a common track (Trains).

Model checking BESs for the model checking problems have been generated using the mCRL2 tools `lps2pbes` and `pbes2bool` with on-the-fly local resolution during BES generation enabled (observe that this is an efficient optimisation, linear in the size of the BES, and is generally used in practice; it is similar to self-loop elimination in parity games). All of the specifications mentioned are used in some model checking experiments.

We list the modal formulae in the syntax of [Koz83] that we have used during our experiments, the precise combinations of formulae and specifications are described in the actual experiment descriptions. The modal formulae in the syntax of the mCRL2 toolset are included in Appendix C.

We have checked for two desired properties in all of our examples: absence of deadlock (5.1) and livelock (5.2). Furthermore there is a number of properties that is applicable to the communication protocols that we have tested. First of all it is desired for communication protocols that a message of a certain type ($d1$) can be received (through $r1$) infinitely often (5.3). The same can be checked for all message types in the specification (5.4). We also check whether it holds that if a receive of message is infinitely often enabled, then it is infinitely often taken (5.5). We also consider some properties of the dining philosophers problem. We do not want philosophers to starve, *i.e.* for all reachable states it should eventually be possible to perform an eat action for all philosophers ($Phil$) (5.6). Also, we want philosophers to have healthy habits; they need to stop eating after a finite amount of time, so they do not stuff themselves (5.7). The last two properties that we consider are typical for simple board games like snake. It is desirable that a game eventually finishes, *i.e.* one of the two players wins (5.8). Of course the game should be fair, hence it must be possible both for player 1 and player 2 to win. We show the formula for player 1, player 2 is analogous (5.9).

$$[\mathsf{true}^*]\langle \mathsf{true}\rangle \mathsf{true} \tag{5.1}$$

$$[\mathsf{true}^*]\mu X.[\tau]X \tag{5.2}$$

$$\nu X.\mu Y.(\langle r1(d1)\rangle X \vee \langle \neg r1(d1)\rangle Y) \tag{5.3}$$

$$\forall d : D.(\nu X.\mu Y.(\langle r1(d)\rangle X \vee \langle \neg r1(d)\rangle Y)) \tag{5.4}$$

$$\forall d : D.([\mathsf{true}^*]\nu X.\mu Y.\nu Z.([r1(d)]X \wedge ([r1(d)]\mathsf{false} \vee [\neg r1(d)]Y) \wedge [\neg r1(d)]Z)) \tag{5.5}$$

$$[\mathsf{true}^*](\forall p : Phil.(\mu Y.([\neg eat(p)]Y \wedge \langle \mathsf{true}\rangle \mathsf{true}))) \tag{5.6}$$

$$\forall p : Phil.(\nu X.\mu Y.[eat(p)]Y \wedge [\neg eat(p)]X) \tag{5.7}$$

$$\mu X.\langle Player1Wins \vee Player2Wins\rangle \mathsf{true} \vee [\mathsf{true}]X \tag{5.8}$$

$$\mu X.\langle Player1Wins\rangle \mathsf{true} \vee \langle \mathsf{true}\rangle X \tag{5.9}$$

The parity games that correspond to these formulae have the following alternation depths. For (5.1), (5.8) and (5.9) the alternation hierarchy is 1. Formula (5.5) has alternation hierarchy 3, and all the others have alternation hierarchy 2.

Equivalence checking The encoding of the branching bisimulation equivalence problem yields parity games with two priorities. The BES from which the parity games have been generated for these problems were generated using the tools `bisimulation` and `pbes2bool`.

The rest of this chapter is structured as follows. Section 5.1 describes the influence of various optimisations to the running time of the solving algorithms. Section 5.2 shows how increasing

the number of priorities affects the running time of the algorithms. A comparison of all parity game algorithms described in Chapter 4 applied to a large number of examples is presented in Section 5.3.

5.1 Practical influence of optimisation techniques

We have investigated the influence of the optimisation techniques described in Section 3.1.1 as follows. We consider the sliding window protocol (SWP) with $|M| = 1$, *i.e.* a single message type, and buffers of size 2. The SWP is a well known network communications protocol; this particular version has the same external behaviour as a buffer.

To determine the influence of optimisation techniques we check the specification for (5.1) deadlock freedom, (5.2) absence of livelock, (5.3) whether a receive of a message is done infinitely often, and (5.5) if a receive of a message is enabled infinitely often, then it is taken infinitely often.

5.1.1 Experiments

For each of the four modal formulae, and for each of the eleven algorithms described in Chapter 4 we have run PGSolver with any possible combination of the optimisations as described in Section 3.1.1, *i.e.* a total number of 5632 experiments. The full set of results is given in Appendix A.

5.1.2 Analysis technique

The large set of numeric data that results from this experiment is hard to interpret in its raw form. As we are looking for the effects of optimisations on the running time of the experiments, we have created scatter plots with on the x-axis the experiment number (determined by the optimisations were enabled). On the y-axis the running time has been set out. Note that runs with the lowest running time, and hence the best performance are found in the top of the graphs. Each graph contains either 128 or 64 points, each of which identifies a run of PGSolver on a specific problem instance (consisting of a single specification with a single modal formula) with the same algorithm, but with different combinations of the optimisations enabled (*i.e.* 2^7 combinations of the 7 optimisations that we have described). In the cases where only 64 measurements are depicted decomposition into strongly connected components was always enabled. Clusters in these graphs are courtesy of certain combinations of optimisations.

We consider an optimisation to be relevant if all runs where the optimisation is disabled show a running time that is significantly longer than the fastest running time.

In the figures, all measurements that have been carried out are marked with $+$, in addition, experiments where a certain optimisation was disabled are marked with X. Exactly which optimisation was disabled is marked in the legend of each graphs.

5.1.3 Results

From the results of our experiments we observe that it is highly beneficial to enable decomposition of the parity game into strongly connected components. A representative image with experimental results is shown in Figure 5.1. This shows all runs, and marks the runs in which SCC decomposition is disabled. An overview of all results for this experiment (with only the optimisations highlighted that we consider in this section) is given in Appendix A.

In all cases where enabling SCC decomposition is not an improvement, either the running time of the experiment is too small ($< 0.5 seconds$) to make any reliable comparison, or enabling SCC decomposition does not significantly degrade performance. Note that in general the parity game that is provided as input consists of a single large SCC, hence SCC decomposition only turns effective once part of the parity game has been solved.

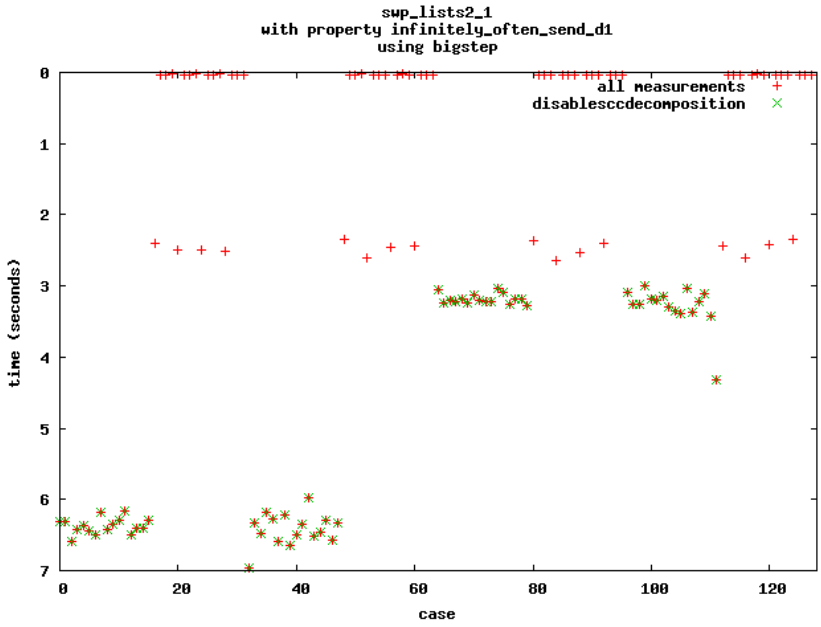


Figure 5.1: Infinitely often receive for SWP, $|M| = 1$, using the bigstep algorithm, visualising effect of SCC decomposition

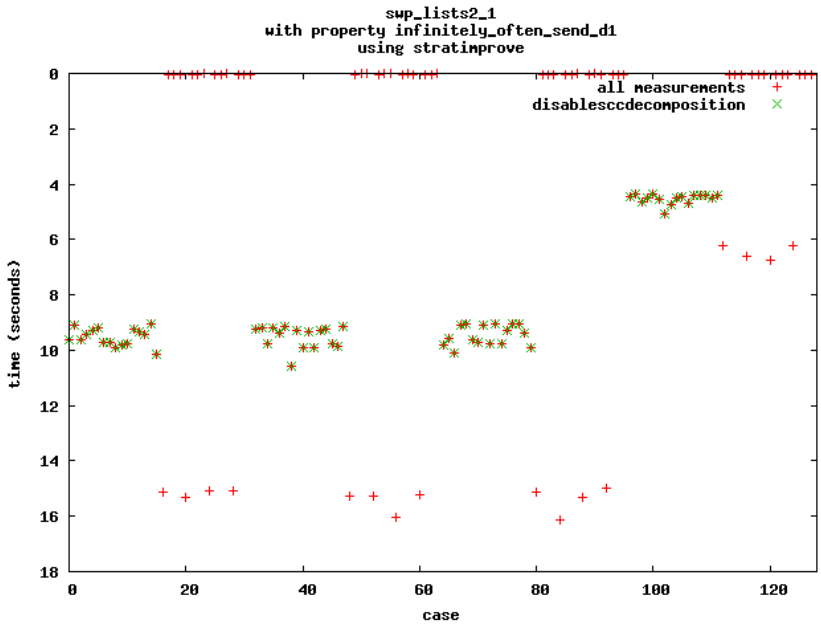


Figure 5.2: Infinitely often receive for SWP, $|M| = 1$, using the strategy improvement algorithm, visualising the effect of SCC decomposition

As can be observed from Figure 5.2, there are some other optimisations that boost the performance of the solver. Because we have seen that SCC decomposition is always beneficial we only consider results in which this optimisation was enabled in the rest of this section.

It turns out that disabling the solving of special (*i.e.* single player and single parity) games is responsible for the other peaks in the running times. A good example in which this can be seen

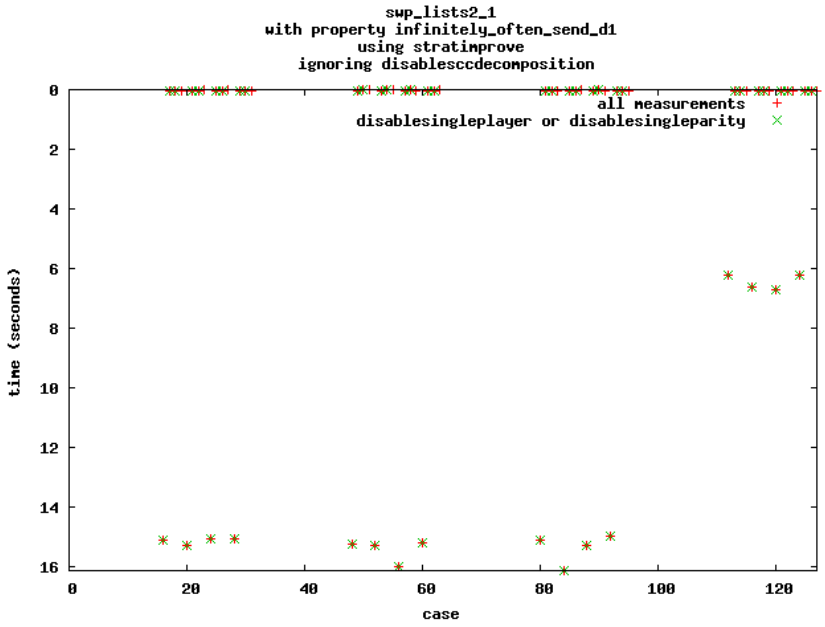


Figure 5.3: Infinitely often receive for SWP, $|M| = 1$, using the strategy improvement algorithm, visualising the effect of solving special games

is Figure 5.3. The full set of results is presented in Appendix A.2. As in all cases where there are any significant differences between running times these are accounted for by the solving of special games, using these optimisations is advisable as well. It is good to note that enabling only one of the two optimisations is not sufficient.

This experiment is by no means exhaustive. To make more reliable conclusions about the effects of the optimisations one would need to extend the experiments. Given that time is limited, and running an extensive series of experiments is extremely time consuming, combined with the fact that we are more interested in the differences between the actual algorithms for solving parity games, and not the optimisations on meta-level as such, we have limited ourselves to the set of experiments described in this section.

We do feel though, that these experiments give a good indication of the optimisations that provide the greatest performance boost in practice. Hence, based on our experiments we consider it good practice to enable decomposition into strongly connected components, as well as the solving of special games.

Note that for solving parity games the optimisation of priority compaction may also be beneficial. However, as we generate the parity games from BESs using the ranks as priority, priority compaction does not alter the priorities, hence from a BES solving point of view priority compaction is not relevant.

5.2 Influence of priorities on the performance of the algorithms

The running time of most parity game algorithms depends on the number of priorities in the parity game, as can be seen in Table 4.1. To show how this manifests itself in practice, we use the model checking problems from the previous section, and alter their priorities; technically we alter the translation of BESs to parity games, by assigning different priorities according to the translation given in the next section.

5.2.1 Experiments

We again use SWP with $|M| = 1$ and buffer size 2. The parity games used in the previous section have been modified such that for each priority there are at most n nodes with that priority, where the experiment is run for $n \in \{1, 2, 3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, \infty\}$ (so 17 different values of n). Note that for smaller maximal block sizes, the number of priorities in the parity game *increases*. The run with $n = \infty$ reflects the original version of the problem, *i.e.* with the ranks used as priority. For all runs SCC decomposition has been enabled, in line with the observations from the previous section. For the other optimisations we have done runs for all 64 remaining combinations.

More formally, given a BES $\mathcal{E}$, we have assign priorities to $X \in \mathsf{bnd}(\mathcal{E})$ according to the function $\mathsf{limrank}_n(X)$, which is defined as $\mathsf{limrank}_n(X) = \mathsf{limrank}_{\nu,X,n}(\mathcal{E}, m)$, where $\mathsf{limrank}_{\nu,X,\mathcal{E}}(n,)$ is inductively defined as follows:

$$\mathsf{limrank}_{\sigma,X,n}(\epsilon, m) = 0$$

$$\mathsf{limrank}_{\sigma,X,n}((\sigma' Y = f)\mathcal{E}, m) = \begin{cases} 0 & \text{if } \sigma = \sigma' \text{ and } X = Y \\ \mathsf{limrank}_{\sigma,X,n}(\mathcal{E}, m+1) & \text{if } \sigma = \sigma' \text{ and } X \neq Y \text{ and } m < n \\ 1 + \mathsf{limrank}_{\sigma,X,n}(\mathcal{E}, 0) & \text{if } \sigma = \sigma' \text{ and } X \neq Y \text{ and } m \geqslant n \\ 1 + \mathsf{limrank}_{\sigma',X,n}((\sigma' Y = f)\mathcal{E}, 0) & \text{if } \sigma \neq \sigma' \end{cases}$$

Let $\mathcal{E}$ be a BES. $\mathsf{limrank}_{\sigma,X,n}(\mathcal{E}, m)$ assigns a priority to X, where n is the maximal number of nodes per priority, m is the number of nodes of the same priority seen so far and σ is the last encountered fixpoint symbol.

5.2.2 Analysis technique

In this section we investigate the change in running time of the algorithms as the number of priorities decreases. For each of the model checking examples we have created a graph for each of the 64 optimisation configurations. In each of these graphs we show the running times for all algorithms for the cases of 1 to 32 nodes per priority, with on the x-axis the maximal number of nodes with the same priority, and on the y-axis the running time. Note that runs with the lowest running time, and hence the best performance are found in the top of the graphs Note that we omit the measurements with a timeout or memory error, as they clutter the graphs, making them hard to interpret. Generally the timeouts occur in the cases of $1 - 4$ nodes per priority.

5.2.3 Results

We first consider the results of the runs in which priority compaction is disabled. We disable priority compaction because it counters the artificial way in which we have assigned priorities, and we expect it to be the only optimisation that does so. Figures 5.4–5.7 show the running times per algorithm, in which the maximal number of vertices with the same priority is restricted to 32. This does not show any connection between the number of priorities and the running times.

It turns out that the optimisations for solving special games interfere with the original algorithms in our example. This is the case because we have restricted ourselves to a very basic specification ($|M| = 1$) in our experiments, where these optimisations are indeed able to solve the problem more efficiently. Figures 5.8–5.11 we show the measurements for solving the parity games without solving special games, and with priority compaction enabled (hence first reducing the number of priorities). This can be compared to Figures 5.12–5.15 where we present the results of the same experiments for the case where solving of special games as well as priority compaction have been disabled. This properly shows the running times of the original algorithms, and the influence of increased numbers of priorities on the performance of the algorithms.

In Figures 5.12–5.15 we see that indeed for all algorithms (except `bigstep`) that have a theoretic running time dependency on the number of priorities d, this also occurs in practice (even when the rest of the structure of the parity game remains the same). Note that the figures are restricted

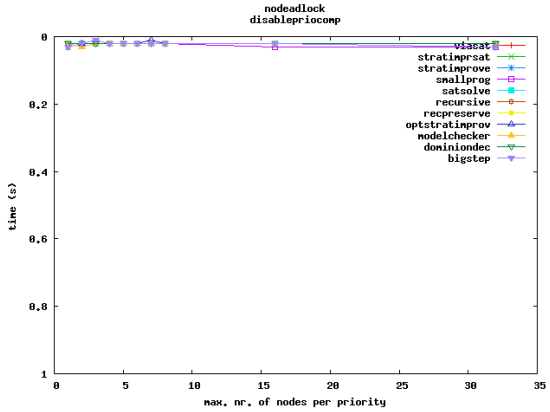


Figure 5.4: No deadlock

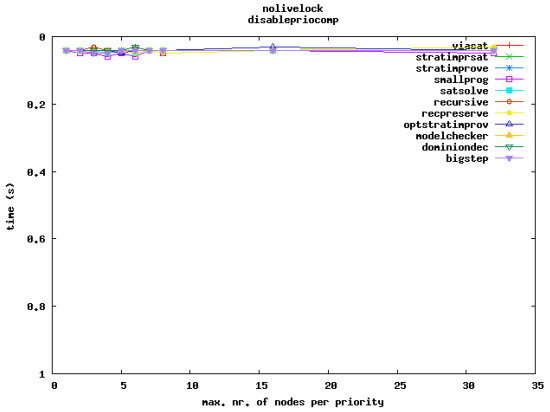


Figure 5.5: No livelock

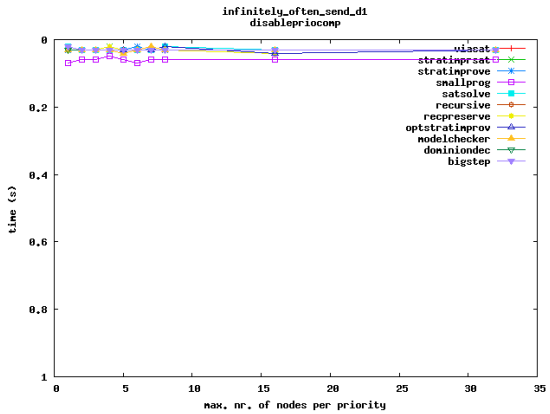


Figure 5.6: Infinitely often sent

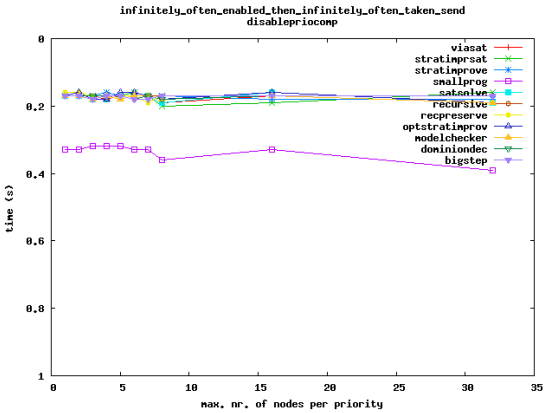


Figure 5.7: Infinitely often enabled then infinitely often taken

to a maximal running time of 200 seconds, and a maximal block size of 32, as these are the areas where the biggest effects are observed. The full set of experiments has running times to 1800 seconds, and maximal block sizes of 2048.

The dependency on d for the `bigstep` algorithm does not show up in our experiments. This can mostly be explained by the fact that the dependence on d in the bigstep algorithm is accounted for by the use of small progress measures as sub-algorithm in dominion decomposition. Hence it is a fair probability that the number of priorities in the sub-game that small progress measures is called for is fairly small anyway. To show that the dependency of the bigstep algorithm on the number of priorities indeed shows in practice, a more in-depth examination of the algorithm with a larger collection of experiments is required.

For small progress measures the effect is not entirely clear from the figures, hence we list the measurements in Table 5.1, to make clear that this algorithm does indeed also show the dependency on d. Recall that the run with $n = \infty$ denotes the parity game generated according to the original rank function. From this table we observe that small progress measures is extremely sensitive to changes in number of priorities when considering formulae with alternation.

The experiments from this section confirm what one might expect from the theoretic running times. All algorithms, except for bigstep, that have a running time dependency on d in theory, also show this dependency in practice. However, in some cases the algorithms for solving special games can be used to prevent running into the dependency on d altogether.

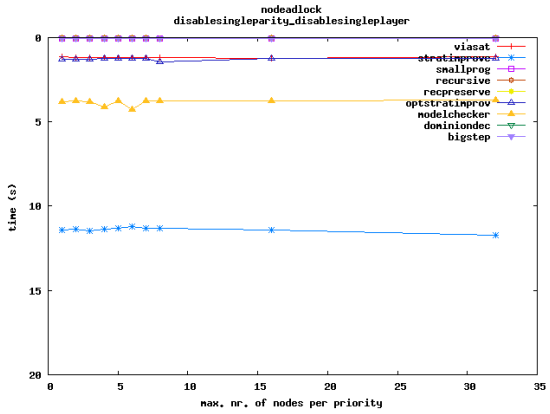


Figure 5.8: No deadlock

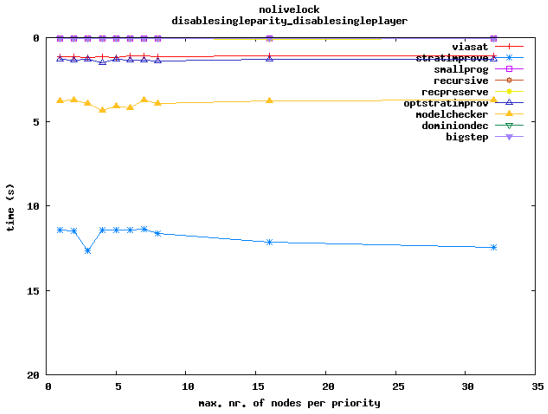


Figure 5.9: No livelock

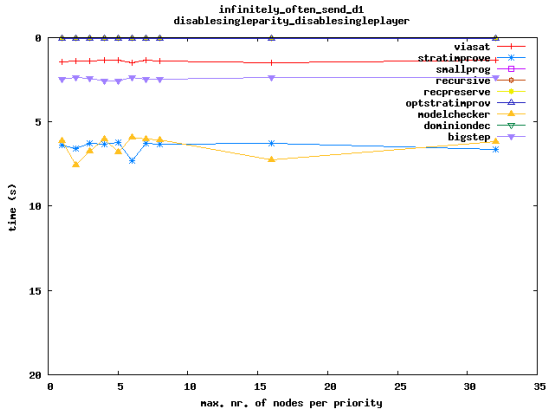


Figure 5.10: Infinitely often sent

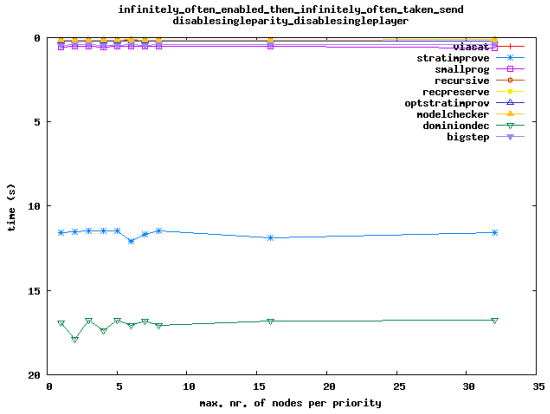


Figure 5.11: Infinitely often enabled then infinitely often taken

Table 5.1: Effect of variable block size for small progress measures

	Solving times			
Max. block size	(5.1)	(5.2)	(5.3)	(5.5)
1	0.03	0.06	t/o	t/o
2	0.03	0.07	t/o	t/o
3	0.04	0.06	t/o	t/o
4	0.04	0.07	t/o	t/o
5	0.03	0.07	t/o	t/o
6	0.04	0.07	t/o	t/o
7	0.03	0.05	t/o	t/o
8	0.04	0.06	t/o	t/o
16	0.03	0.07	t/o	t/o
32	0.03	0.06	t/o	t/o
64	0.03	0.07	t/o	t/o
128	0.03	0.06	t/o	593.40
256	0.03	0.07	t/o	8.28
512	0.03	0.06	t/o	1.22
1014	0.03	0.06	t/o	0.69
2048	0.03	0.06	t/o	0.56
∞	0.03	0.06	30.34	0.52

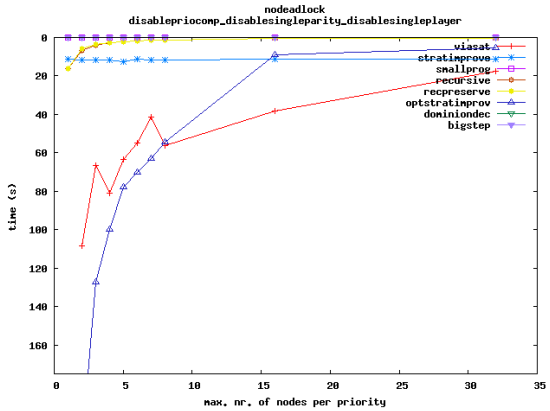


Figure 5.12: No deadlock

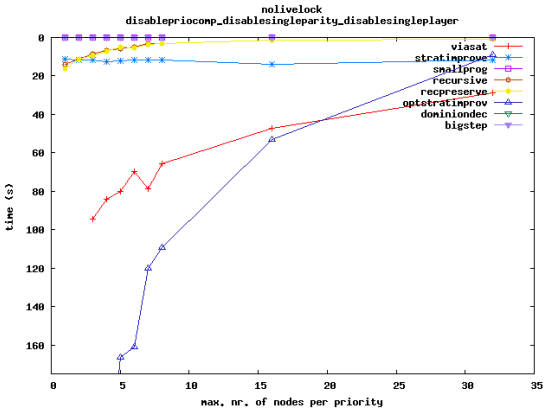


Figure 5.13: No livelock

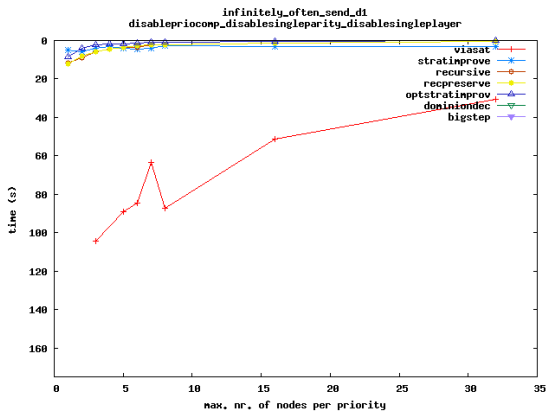


Figure 5.14: Infinitely often sent

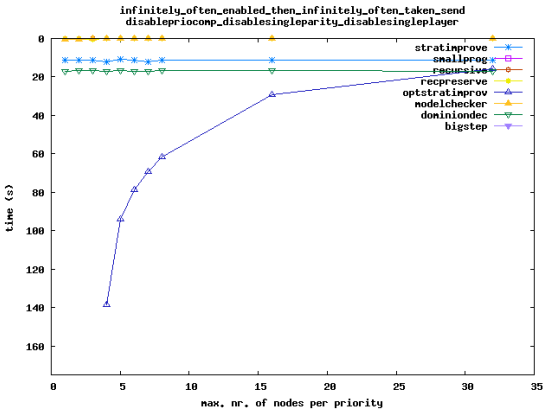


Figure 5.15: Infinitely often enabled then infinitely often taken

5.3 Comparison of parity game algorithms

In this section we compare the algorithms described in Chapter 4 with respect to their performance in practical model checking cases. Therefore we have taken a large number of model checking problems from the examples in the mCRL2 toolset. The experiments are carried out on the parity game versions of these problems, obtained through the mentioned encodings.

5.3.1 Experiments

Based on the results from the previous sections, we run all algorithms with the default optimisations supplied by PGSolver. The most important reason for allowing the optimisations to be carried out is that we want to get an indication of the performance of the algorithm for practical model checking purposes. We are aware that the optimisations that are used might improve the actual running times of the algorithms in some examples. In the rest of this section we provide and analyse the results of a total of 2783 experiments.

Model checking problems We consider the examples from the introduction of this chapter, in combination with the modal formulae described there. In Table 5.2 we give an overview of the properties that we checked, and for which specifications these properties were checked.

Equivalence checking As input to the equivalence checking problem we used four descriptions of well-studied communications protocols, *viz.*, the one-place buffer (OPB), two variations of the Alternating Bit Protocol (ABP) and the Concurrent Alternating Bit Protocol (CABP). For each protocol we varied the size of the set of messages M that could be exchanged.

5.3.2 Analysis techniques

Because of the large number of experiments that we consider in this section, and the number of different cases that we consider, we use a tabulated presentation of the measurements. Each of the tables considers one specification, and lists for each of the properties, and each of the algorithms, the time required for executing PGSolver with the algorithm. This allows for a relatively quick inventarisation of the best and worst algorithms in each of the cases. Additionally we have created tools to compute the number of times that an algorithm is among the best an worst performing algorithms in order to get a better overall view. In the tables that we present, t/o denotes a timeout of the corresponding run; the time required exceeds 30 minutes. Furthermore ME denotes termination of PGSolver with a memory error.

Table 5.2: Combinations of modal formulae and specifications

Formula	Specifications
No deadlock (5.1)	All
No livelock (5.2)	All
Infinitely often receive $d1$ (5.3)	ABP, CABP, SWP
Infinitely often receive all d (5.4)	ABP, CABP, SWP
Infinitely often enabled then infinitely often taken receive (5.5)	ABP, CABP, SWP
No starvation (5.6)	Dining
No stuffing (5.7)	Dining
Eventually player 1 or player 2 (black or white) wins (5.8)	Domineering, Snake
Player 1/Player 2 (Black/White) can win (5.9)	Domineering, Snake

5.3.3 Results

The full set of results corresponding to the model checking experiments are presented in Appendix B.1, the equivalence checking results can be found in Appendix B.2. In Table 5.3 the results are shown for the SWP with buffer size two, and varying numbers of messages. The results that we see here are representative for all measurements for model checking that we have collected. Table 5.4 shows the results of equivalence checking experiments in which one of the two processes involved was CABP. These results are representative for the equivalence checking experiments.

Table 5.3: SWP, buffer size 2

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
						nodeadlock						
2	71,094	0.29	0.30	0.30	0.34	0.30	0.30	0.29	0.29	0.30	0.30	0.29
3	269,286	1.38	1.38	1.37	1.47	1.38	1.38	1.37	1.38	1.38	1.38	1.39
4	728,390	3.82	3.82	3.80	4.34	3.83	3.88	3.83	3.81	3.80	3.80	3.87
5	1,614,606	10.77	8.72	8.73	9.95	8.76	8.73	8.71	8.79	8.70	8.80	8.73
6	3,135,606	19.64	19.91	21.03	24.70	19.95	19.79	21.41	20.18	21.41	22.04	20.36
7	5,540,534	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	9,120,006	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
						nolivelock						
2	113,286	0.51	0.50	0.50	0.62	0.50	0.52	0.50	0.52	0.51	0.49	0.51
3	426,426	2.14	2.03	2.13	2.66	2.12	2.12	2.13	2.13	2.13	2.13	2.12
4	1,149,446	6.02	6.01	5.77	6.94	5.76	5.77	5.73	5.75	5.79	5.74	5.79
5	2,542,506	13.73	13.71	13.67	16.56	13.74	15.42	13.76	13.75	13.71	13.73	13.71
6	4,930,566	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
7	8,703,386	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	14,315,526	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
						infinitely often receive d1						
2	78,584	0.37	0.36	0.35	0.67	0.37	0.37	0.37	0.37	0.36	0.37	0.36
3	290,672	1.58	1.52	1.53	2.91	1.57	1.55	1.54	1.53	1.55	1.55	1.54
4	774,472	4.21	4.21	4.19	7.73	4.21	4.33	4.20	4.20	4.19	4.21	4.19
5	1,699,208	9.68	9.74	10.97	19.79	9.74	9.65	11.15	10.95	9.72	9.70	9.70
6	3,275,576	24.01	24.07	22.13	41.77	23.18	23.14	22.28	22.59	22.22	22.64	22.28
7	5,755,744	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	9,433,352	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
						infinitely often receive for all d						
2	157,165	0.80	0.79	0.80	1.31	0.80	0.81	0.81	0.81	0.81	0.81	0.81
3	872,008	4.83	4.83	4.83	9.58	4.98	4.88	4.83	4.72	5.01	4.90	4.83
4	3,097,875	19.04	19.13	19.01	31.21	18.87	19.13	18.97	19.09	18.98	19.96	19.01
5	8,496,022	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
						infinitely often enabled then infinitely often taken receive						
2	457,833	13.10	13.42	13.55	27.90	12.75	11.76	12.05	11.69	12.17	11.75	11.83
3	2,574,406	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
4	9,223,947	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
5	25,444,512	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

The observation that stands out most from these results is that small progress measures is the slowest algorithm throughout the experiments. Even for large instances of model checking problems (see *e.g.* SWP with window size 2, with $|M| = 5$) it seems that the overhead of the algorithm is too large for the better theoretic complexity of the algorithm to pose an improvement in practice.

In addition to the bad performance of small progress measures there are two interesting cases in the equivalence checking problem, *viz.* ABP - CABP and ABP - OPB. We see that in these cases some algorithms do not terminate within 30 minutes, or terminate with memory errors. [4] Other algorithms however terminate within seconds on the same problem instances. One would

[4]The memory errors are specific to the used version of the PGSolver toolset. A development version obtained through personal communication with Oliver Friedmann does not suffer from these issues, with that version however, all cases that show a memory error time out.

Table 5.4: Branching bisimilarity with CABP

		Solving times (bigstep)										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					ABP_1 – CABP							
2	205,846	t/o	ME	ME	1.97	t/o	t/o	1.53	2.15	ME	1.81	ME
3	334,038	t/o	ME	ME	3.61	t/o	t/o	2.59	3.89	ME	3.06	ME
4	479,158	t/o	ME	ME	5.27	t/o	t/o	3.83	5.36	ME	4.37	ME
5	641,206	ME	ME	ME	8.88	t/o	t/o	6.34	7.27	t/o	5.93	ME
6	820,182	ME	ME	ME	10.30	t/o	t/o	6.77	11.70	ME	7.77	ME
7	1,016,086	ME	ME	ME	11.08	t/o	t/o	8.78	16.03	ME	11.03	ME
8	1,228,918	ME	ME	ME	13.92	ME	t/o	10.26	14.32	ME	11.91	ME
9	1,458,678	ME	ME	ME	16.08	ME	t/o	12.58	17.85	t/o	14.81	ME
10	1,705,366	ME	ME	ME	19.00	ME	t/o	15.26	21.19	ME	17.61	ME
16	3,540,982	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
20	5,103,286	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
24	6,936,438	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
28	9,040,438	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
32	11,415,286	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
					ABP_2 – CABP							
2	36,492	0.13	0.14	0.14	0.20	0.14	0.13	0.13	0.14	0.13	0.13	0.14
3	55,149	0.22	0.22	0.21	0.38	0.22	0.22	0.21	0.21	0.22	0.22	0.22
4	74,094	0.30	0.30	0.29	0.44	0.29	0.30	0.29	0.29	0.30	0.30	0.29
5	218,352	1.24	8.04	8.19	1.38	1.17	1.29	1.29	1.29	1.29	1.29	1.15
6	264,414	1.48	9.61	9.17	1.93	1.55	1.51	1.37	1.37	1.53	1.52	1.52
7	311,276	1.84	11.07	10.75	2.03	1.74	1.71	1.72	1.69	1.72	1.71	1.71
8	152,754	0.68	0.67	0.67	1.02	0.68	0.67	0.68	0.68	0.68	0.68	0.67
9	173,139	0.77	0.76	0.76	1.17	0.75	0.76	0.76	0.76	0.75	0.76	0.76
10	193,812	0.91	0.90	0.90	1.38	0.95	0.91	0.95	0.91	0.91	0.90	0.90
16	323,898	1.51	1.50	1.50	2.61	1.50	1.51	1.49	1.50	1.50	1.51	1.53
20	993,282	6.06	31.56	32.26	7.00	5.62	5.70	5.53	5.54	5.55	5.52	5.51
24	1,230,330	8.70	39.57	40.12	9.01	8.23	7.57	7.85	7.53	7.44	7.34	7.45
28	1,480,178	11.05	47.22	49.41	11.71	9.74	8.82	9.63	9.72	8.80	9.68	9.61
32	1,742,826	11.46	53.56	55.56	13.19	10.81	10.71	10.65	10.93	10.71	10.51	11.04
					CABP – OPB							
2	12,022	0.04	0.04	0.04	0.05	0.04	0.03	0.04	0.03	0.04	0.04	0.04
3	21,470	0.08	0.08	0.08	0.10	0.07	0.07	0.07	0.07	0.08	0.08	0.08
4	33,270	0.14	0.14	0.13	0.15	0.13	0.13	0.14	0.13	0.13	0.13	0.13
5	47,422	0.19	0.22	0.21	0.26	0.20	0.22	0.22	0.21	0.20	0.21	0.22
6	63,926	0.30	0.31	0.32	0.33	0.31	0.30	0.28	0.31	0.28	0.31	0.28
7	82,782	0.37	0.37	0.37	0.43	0.37	0.36	0.37	0.37	0.39	0.37	0.37
8	103,990	0.48	0.49	0.49	0.57	0.48	0.48	0.49	0.49	0.48	0.49	0.48
9	127,550	0.60	0.61	0.66	0.71	0.61	0.61	0.61	0.61	0.60	0.61	0.61
10	153,462	0.73	0.73	0.74	0.86	0.73	0.74	0.74	0.73	0.73	0.74	0.73
16	358,326	1.78	1.77	1.86	2.31	1.77	1.77	1.78	1.80	1.78	1.78	1.77
20	541,942	2.81	2.81	2.80	3.22	2.79	2.88	2.78	2.84	2.81	2.78	2.84
24	763,190	4.26	4.27	4.12	4.84	4.12	4.19	4.19	4.05	4.11	4.33	4.00
28	1,022,070	5.56	6.09	6.15	7.07	6.11	5.47	6.10	6.06	6.11	6.16	5.53
32	1,318,582	7.89	7.82	7.66	8.69	7.72	7.51	7.81	7.55	7.54	7.60	7.58

expect that the BES for branching bisimilarity of both variants of ABP with CABP and OPB would behave similarly, hence we have verified these results by repeating the experiment, showing similar results. The most important difference between both parity games is the difference in size, which can be accounted for by the different structure of the processes ABP_1 and ABP_2.

In order to get a better insight in the general performance of the algorithms, we show for each algorithm the number of times that algorithm is amongst the top 10 percent, and the slowest 10 percent of the runs in a problem instance. These results are shown in Table 5.5. Note that we always count results that are within 0.1 seconds of the slowest/fastest run.

From these results we observe that except for small progress measures, all algorithms are relatively close with respect to the number of cases in which they perform extremely good or bad. When looking at the absolute numbers, the recursive algorithm is the one that is among the best 10% most often, and among the worst 10% least often, hence we may consider this the most efficient algorithm based on our experiments.

Table 5.5: Number of times that algorithms are within 10 percent of the best/worst algorithm

Strategy	# top 10%	# worst 10%
viasat	278	216
stratimprsat	285	216
satsolve	282	215
smallprog	198	296
stratimprove	286	214
optstratimprov	283	218
recursive	292	212
recpreserve	285	219
dominiondec	291	215
bigstep	277	220
modelchecker	286	215

An interesting observation that we may make based upon our experiments, is that the recursive algorithm scores better than the recursive algorithm with preservation, whereas the latter is supposed to be an optimisation of the first. A similar effect occurs when comparing the strategy improvement algorithm with the optimal strategy improvement algorithm. For the strategy improvement, this difference contrasts with the results by Schewe [Sch08], where results are presented that show that the optimal strategy improvement by far outperforms the strategy improvement algorithm. It turns out that the difference in running times is affected by the optimisations that were enabled for our experiments, of which we expect that similar optimisations were not used by Schewe, but which have been shown to have dramatic effects on the running time in Section 5.1. Table 5.6 shows the differences between having all optimisations enabled and having all optimisations disabled, and indeed displays differences similar to the results in [Sch08] in case optimisations are disabled. Additional differences could be caused by our choice of examples, as we are interested in realistic model checking problems, compared to the random games that are presented in [Sch08].

Table 5.6: SWP with window size 2 and $|M| = 1$, measurements from strategy improvement and optimal strategy improvement, where optimised means that all optimisations were enabled, and unoptimised means no optimisations were enabled.

Formula	Optimised		Unoptimised	
	stratimprov	optstratimprov	stratimprov	optstratimprov
5.1	0.02	0.02	23.01	1.13
5.2	0.04	0.04	92.60	2.14
5.3	0.03	0.03	9.90	0.06
5.5	0.27	0.26	276.73	3.81

5.4 Conclusions

We assume that it is desired that algorithms do not show extremely bad behaviour in practice, when there are other algorithms that do perform fairly well. This leaves only the recursive algorithm, recursive with preservation and bigstep that should be considered for practical use. Based on our results we recommend further investigation of the use of the recursive and bigstep algorithms for solving BESs.

Furthermore we have seen that it pays to try to reduce the number priorities in the input, as

a large number of priorities adversely affects the time required for solving. Likewise it pays to enable optimisations like decomposition into strongly connected components, as well as employing optimal algorithms for solving special cases.

Chapter 6

Small progress measures for Boolean Equation Systems

Despite the results in the previous section, where it was shown that small progress measures [Jur00] is generally not the most efficient algorithm for solving parity games, we investigate the translation of the algorithm to the BES framework. As small progress measures is used as part of the bigstep algorithm, which was shown to be one of the most efficient algorithms, this chapter is a first step towards applying the bigstep algorithm to BESs.

The work in this chapter also serves as a proof of concept to show that parity game algorithms can be applied to BESs in SRF almost immediately. This opens up the way of solving BESs using parity game algorithms without explicitly translating the BES. In addition we show that the algorithm can be generalised to BESs with arbitrary right hand sides, hence also eliminating the translation to SRF. Note that a parallel implementation of this algorithm has been presented by Van de Pol and Weber [PW08]; this opens up the way to solving BESs in parallel.

6.1 Small progress measures

The small progress measures algorithm utilises a decoration of the game graph. Each vertex is attributed with a tuple with as length the maximal priority occurring in the parity game. Initially this is the tuple $\overline{0}$ with 0 in all positions. Furthermore, all even positions always remain 0, and odd positions i are limited to the number of vertices with priority i. On these tuples a lexicographic ordering is defined such that $(n_0, n_1, \ldots, n_k) \equiv_i (m_0, m_1, \ldots, m_l)$ if and only if $(n_0, n_1, \ldots, n_i) \equiv (m_0, m_1, \ldots, m_i)$ with $\equiv \in \{<, \leqslant, =, \geqslant, >\}$. If $k < i$ or $l < i$ positions n_j with $k \leqslant j \leqslant i$ and m_j with $l \leqslant j \leqslant i$ are assumed to be 0. The following example illustrates the orderings.

Example 6.1.1. $(0,1,0,1) =_0 (0,2,0,1)$ is equivalent to $(0) = (0)$ and hence is true. $(0,1,0,1) <_1 (0,2,0,1)$ is equivalent to $(0,1) < (0,2)$ and hence is also true, whereas $(0,1,0,1) \geqslant_3 (0,2,0,1)$ is $(0,1,0,1) \geqslant (0,2,0,1)$ is false.

Definition 6.1.2. A function $\varrho{:}V \to \mathbb{N}^d$, with $d = \max(\{p(v) \mid v \in V\})$ the maximal priority in the game, is a *parity progress measure* for parity graph $\mathcal{G} = (V, E, p)$ if for all $(v, w) \in E$ we have

$$\begin{cases} \varrho(v) \geqslant_{p(v)} \varrho(w) & \text{if } p(v) \text{ is even} \\ \varrho(v) >_{p(v)} \varrho(w) & \text{if } p(v) \text{ is odd} \end{cases}$$

Consider game graph $\mathcal{G} = (V, E, p{:}V \to \mathbb{N})$. For every $i \in \mathbb{N}$ we denote with $V_i \subseteq V$ the set of vertices in $\mathcal{G}$ with priority i. Furthermore we let $n_i = |V_i|$, the number of vertices with priority i.

We define the finite subset M of $\mathbb{N}^d$, such that it is the finite set of d-tuples with zeros on even positions (counting from 0), and non-negative integers bound by n_i on odd positions i as follows:

$$M = \begin{cases} [1] \times [n_1 + 1] \times [1] \times [n_3 + 1] \times \cdots \times [1] \times [n_{d-1} + 1] & \text{if } d \text{ is even} \\ [1] \times [n_1 + 1] \times [1] \times [n_3 + 1] \times \cdots \times [1] \times [n_{d-2} + 1] \times [1] & \text{if } d \text{ is odd} \end{cases}$$

In which $[n]$ is defined formally as $[n] \triangleq \{i \mid 0 \leqslant i < n\}$.

Theorem 6.1.3. [Jur00] There is a parity progress measure $\varrho{:}V \to M$ for parity graph $\mathcal{G}$ if and only if all cycles in $\mathcal{G}$ are even.

The parity progress measure is still too restrictive to be used for computing a winning strategy, as it does not allow for the expression of odd cycles, hence we add a largest element $\top$ to M, such that $M^\top = M \cup \{\top\}$. Given $\varrho{:}V \to M^\top$ and $(v, w) \in E$ then $\mathsf{Prog}(\varrho, v, w)$ is the least (w.r.t. $\leqslant$) $m \in M^\top$ such that

$$\begin{cases} m \geqslant_{p(v)} \varrho(w) & \text{if } p(v) \text{ is even} \\ m >_{p(v)} \varrho(w), \text{ or } m = \varrho(w) = \top & \text{if } p(v) \text{ is odd} \end{cases}$$

A function $\varrho{:}V \to M^\top$ is a game parity progress measure if and only if for all $v \in V$:

- if $v \in V_{Even}$ then $\exists_{(v,w)\in E} \varrho(v) \geqslant_{p(v)} \mathsf{Prog}(\varrho, v, w)$, i.e. $\varrho(v) \geqslant_{p(v)} \min\{(v, w) \in E \mid \mathsf{Prog}(\varrho, v, w)\}$
- if $v \in V_{Odd}$ then $\forall_{(v,w)\in E} \varrho(v) \geqslant_{p(v)} \mathsf{Prog}(\varrho, v, w)$, i.e. $\varrho(v) \geqslant_{p(v)} \max\{(v, w) \in E \mid \mathsf{Prog}(\varrho, v, w)\}$

Given a game parity progress measure ϱ, we define strategy $\psi_{Even}{:}V_{Even} \to V$ for player *Even* such that for all $v \in V_{Even}$, $\psi_{Even}(v) = u$, with $\varrho(u) = \min\{\varrho(w) \mid vEw\}$. In words, ψ_{Even} is the successor u of v which minimises $\varrho(u)$. The winning set $||\varrho||$ is the set $\{v \mid v \in V \text{ and } \varrho(v) \neq \top\}$.

It was proven by Jurdziński that strategy ψ_{Even} computed from game parity progress measure ϱ is a winning strategy for player *Even* from $||\varrho||$ [Jur00]. Furthermore it was shown that there is a game parity progress measure $\varrho{:}V \to M^\top$ such that $||\varrho||$ is the winning set of player *Even*.

Algorithm We repeat the algorithm for solving parity games based on small progress measures as presented by Jurdziński [Jur00]. Let $\sqsubseteq$ be an ordering on the set of functions of type $V \to M^\top$. Given functions $\mu, \varrho{:}V \to M^\top$, $\mu \sqsubseteq \varrho$ if and only iff $\mu(v) \leqslant \varrho(v)$ for all $v \in V$. As we are dealing with finite graphs, the set of functions of type $V \to M^\top$ is finite. Furthermore there are greatest and least elements, hence $\sqsubseteq$ defines a complete lattice. We use $\sqsubset$ if $\mu \sqsubseteq \varrho$ and $\mu \neq \varrho$.

The algorithm uses a family of $\mathsf{Lift}(\_, v)(\_)$ operators on $V \to M^\top$ for all $v \in V$. $\mathsf{Lift}(\varrho, v)(\_)$ is defined as follows:

$$\mathsf{Lift}(\varrho, v)(u) = \begin{cases} \varrho(u) & \text{if } u \neq v \\ \min_{(v,w)\in E} \mathsf{Prog}(\varrho, v, w) & \text{if } u = v \in V_{Even} \\ \max_{(v,w)\in E} \mathsf{Prog}(\varrho, v, w) & \text{if } u = v \in V_{Odd} \end{cases}$$

Observe that for every $v \in V$, $\mathsf{Lift}(\_, v)(\_)$ is $\sqsubseteq$-monotonous, i.e. for $\varrho_1 \sqsubseteq \varrho_2$, $\mathsf{Lift}(\varrho_1, v)(\_) \sqsubseteq \mathsf{Lift}(\varrho_2, v)(\_)$. Furthermore, a game parity progress measure can be computed by determining the simultaneous fixpoint of all $\mathsf{Lift}(\_, v)(\_)$ operators. This leads to Algorithm 1 by Jurdziński [Jur00].

Given parity game $\Gamma = (V, E, p{:}V \to \mathbb{N}, (V_{Even}, V_{Odd}))$. Algorithm 1 computes winning sets for both players and a winning strategy for player *Even* from his winning set in $O(d \cdot e \cdot \frac{n}{d}^{d/2})$ time and $O(d \cdot n \cdot \log n)$ space.

Algorithm 1 ProgressMeasureLifting [Jur00]

$\mu \leftarrow \lambda v \in V.\overline{0}$
while $\mu \sqsubset \lambda u \in V.\mathsf{Lift}(\mu, v)(u)$ for some $v \in V$ **do**
  $\mu := \lambda u \in V.\mathsf{Lift}(\mu, v)(u)$
end while

6.2 Progress measures on Boolean Equation Systems

We show that the small progress measures algorithm can also be adjusted in such a way that it can directly be applied to BESs in SRF. In addition we show that we can generalise the algorithm such that it can be applied to BESs with arbitrary right hand sides.

6.2.1 Progress measures on Boolean Equation Systems in SRF

Given the equivalence of BESs and parity games from Section 3.3.2, we can apply Algorithm 1 directly to a closed BES $\mathcal{E}$ in SRF. Instead of first translating $\mathcal{E}$ to a parity game, we use $\mathcal{E}$s dependency graph $\mathcal{G}_\mathcal{E}$. We omit the proofs in this section, as they follow directly from the correspondence between BESs and parity games, and the corresponding results in Section 6.1.

The algorithm we present attaches to each equation a tuple with as length the maximal rank occurring in the BES. Initially this is the tuple $\overline{0}$. Furthermore all even positions always remain 0, and odd positions i are limited to the number of equations in the block with rank i. We use the same lexicographic ordering as in Section 6.1.

Definition 6.2.1. A function $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to \mathbb{N}^{\mathsf{ah}(\mathcal{E})}$ is a parity progress measure for BES $\mathcal{E}$ in SRF if for all $X \in \mathsf{bnd}(\mathcal{E})$ and $X_i \in \mathsf{occ}(\mathsf{rhs}(X, \mathcal{E}))$ we have

$$\begin{cases} \varrho(X) \geqslant_{\mathsf{rank}(X)} \varrho(X_i) & \text{if } \mathsf{rank}(X) \text{ is even} \\ \varrho(X) >_{\mathsf{rank}(X)} \varrho(X_i) & \text{if } \mathsf{rank}(X) \text{ is odd} \end{cases}$$

Consider a BES $\mathcal{E}$ in SRF. For every $i \in \mathbb{N}$ we denote with $\mathcal{E}_i$ the block of equations in $\mathcal{E}$ with rank i, i.e. $\mathcal{E}_i \triangleq \{\sigma X = f \mid \mathsf{rank}(X) = i\}$. We let $n_i = |\mathcal{E}_i|$, the number of equations with rank i. We define finite subset M of $\mathbb{N}^{\mathsf{ah}(\mathcal{E})}$, such that it is the finite subset of $\mathsf{ah}(\mathcal{E})$-tuples with zeros on even positions, and non-negative integers bound by n_i on odd positions i as follows:

$$M = \begin{cases} [1] \times [n_1 + 1] \times [1] \times [n_3 + 1] \times \cdots \times [1] \times [n_{\mathsf{ah}(\mathcal{E})-1} + 1] & \text{if } \mathsf{ah}(\mathcal{E}) \text{ is even} \\ [1] \times [n_1 + 1] \times [1] \times [n_3 + 1] \times \cdots \times [1] \times [n_{\mathsf{ah}(\mathcal{E})-2} + 1] \times [1] & \text{if } \mathsf{ah}(\mathcal{E}) \text{ is odd} \end{cases}$$

Theorem 6.2.2. There is a parity progress measure for BES $\mathcal{E}$ in SRF if and only if all cycles in the dependency graph of $\mathcal{E}$ are ν-dominated.

As before we extend the parity progress measure by adding largest element $\top$ to M, such that $M^\top = M \cup \{\top\}$. Given $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$, $X \in \mathsf{bnd}(\mathcal{E})$ and $X_i \in \mathsf{occ}(\mathsf{rhs}(X, \mathcal{E}))$, then $\mathsf{Prog}(\varrho, X, X_i)$ is the least $m \in M^\top$ such that

$$\begin{cases} m \geqslant_{\mathsf{rank}(X)} \varrho(X_i) & \text{if } \mathsf{rank}(X) \text{ is even} \\ m >_{\mathsf{rank}(X)} \varrho(X_i), \text{ or } m = \varrho(X_i) = \top & \text{if } \mathsf{rank}(X) \text{ is odd} \end{cases}$$

A function $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$ is a BES parity progress measure if and only if for all $X \in \mathsf{bnd}(\mathcal{E})$:

- if $\mathsf{rank}(X)$ is even then $\exists_{Y \in \mathsf{occ}(\mathsf{rhs}(X,\mathcal{E}))} \varrho(X) \geqslant_{\mathsf{rank}(X)} \mathsf{Prog}(\varrho, X, Y)$, i.e. $\varrho(X) \geqslant_{\mathsf{rank}(X)} \min\{Y \in \mathsf{occ}(\mathsf{rhs}(X, \mathcal{E})) \mid \mathsf{Prog}(\varrho, X, Y)\}$
- if $\mathsf{rank}(X)$ is odd then $\forall_{Y \in \mathsf{occ}(\mathsf{rhs}(X,\mathcal{E}))} \varrho(X) \geqslant_{\mathsf{rank}(X)} \mathsf{Prog}(\varrho, X, Y)$, i.e. $\varrho(X) \geqslant_{\mathsf{rank}(X)} \max\{Y \in \mathsf{occ}(\mathsf{rhs}(X, \mathcal{E})) \mid \mathsf{Prog}(\varrho, X, Y)\}$

Observe that the notion of strategy in the parity game approach corresponds to a witness for truth of a disjunctive equation in the domain of BES. Intuitively, for every propositional variable X with $\mathsf{op}(X) = \vee$ and solution true, the witness determines a disjunct that makes the valuation of X true.

Given $\mathcal{X}_\nu = \{X \mid X \in \mathsf{bnd}(E) \text{ and } \mathsf{rank}(X) \text{ is even }\}$, we define witness $\omega_\nu{:}\mathcal{X}_\nu \to \mathsf{bnd}(\mathcal{E})$ for all equations in X in $\mathcal{X}_\nu$ such that $\omega_\nu(X) = Y$, with $\varrho(Y) = \min\{\varrho(Z) \mid Z \in \mathsf{occ}(\mathsf{rhs}(X, \mathcal{E}))\}$. In other words, $\omega_\nu(X)$ is the variable Y in the right hand side of X which minimises $\varrho(Y)$.

$||\varrho|| = \{X | X \in \mathsf{bnd}(\mathcal{E}) \text{ and } \varrho(X) \neq \top\}$ is the set of variables having solution true.

The following two properties follow immediately.

Corollary 6.2.3. ω_ν computed from BES parity progress measure ϱ is a witness for the truth of each X in $||\varrho||$.

Corollary 6.2.4. Let $\mathcal{E}$ be a BES in SRF. There is a BES parity progress measure $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$ such that $||\varrho||$ is the set of variables that are true.

Algorithm We define an ordering $\sqsubseteq$ on the set of functions of type $\mathsf{bnd}(\mathcal{E}) \to M^\top$. Given functions $\mu, \varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$, $\mu \sqsubseteq \varrho$ if and only if $\mu(X) \leqslant \varrho(X)$ for all $X \in \mathsf{bnd}(\mathcal{E})$. Also we are dealing with finite Boolean Equation Systems, hence the set of functions of type $\mathsf{bnd}(\mathcal{E}) \to M^\top$ is finite. Additionally there are greatest and least elements, hence $\sqsubseteq$ defines a complete lattice.

The algorithm uses a family of $\mathsf{Lift}(\_, X)(\_)$ operators on $\mathsf{bnd}(\mathcal{E}) \to M^\top$ for all $X \in \mathsf{bnd}(\mathcal{E})$ and $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$. $\mathsf{Lift}(\varrho, X)(\_)$, for $X \in \mathsf{bnd}(\mathcal{E})$ is defined as follows:

$$\mathsf{Lift}(\varrho, X)(Y) = \begin{cases} \varrho(Y) & \text{if } X \neq Y \\ \min_{Z \in \mathsf{occ}(\mathsf{rhs}(X,\mathcal{E}))} \mathsf{Prog}(\varrho, X, Z) & \text{if } X = Y \text{ and } \mathsf{op}(X) \neq \wedge \\ \max_{Z \in \mathsf{occ}(\mathsf{rhs}(X,\mathcal{E}))} \mathsf{Prog}(\varrho, X, Z) & \text{if } X = Y \text{ and } \mathsf{op}(X) = \wedge \end{cases}$$

As before, for every $X \in \mathsf{bnd}(\mathcal{E})$, $\mathsf{Lift}(\_, X)(\_)$ is $\sqsubseteq$-monotonous. A BES parity progress measure can be computed by determining the simultaneous fixpoint of all $\mathsf{Lift}(\_, X)(\_)$ operators. This results in Algorithm 2.

Algorithm 2 BESProgressMeasureLifting

$\mu \leftarrow \lambda X \in \mathsf{bnd}(\mathcal{E}).\overline{0}$
while $\mu \sqsubset \lambda Y \in \mathsf{bnd}(\mathcal{E}).\mathsf{Lift}(\mu, X)(Y)$ for some $X \in \mathsf{bnd}(\mathcal{E})$ **do**
  $\mu \leftarrow \lambda Y \in \mathsf{bnd}(\mathcal{E}).\mathsf{Lift}(\mu, X)(Y)$
end while

Given the algorithm for parity games and the similarity between BESs and parity games we find that Algorithm 2 computes the set of variables that is true in $O(\mathsf{ah}(\mathcal{E}) \cdot e \cdot \frac{n}{\mathsf{ah}(\mathcal{E})}^{\mathsf{ah}(\mathcal{E})/2})$, where n is the number of equations and e is the cumulative size of the right hand sides in $\mathcal{E}$.

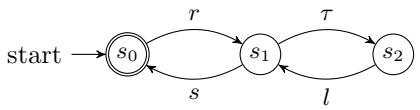
6.2.2 Progress measures for Boolean Equation Systems in RF

The algorithm as presented in the previous section only works for BESs in SRF. The definitions that are used can however be generalised elegantly such that the algorithm can be applied to BESs in RF.

This generalisation enables the solving of arbitrary BESs without the penalty of the linear blow-up needed for transformation to SRF. This is useful for BES in which conjuncts and disjuncts occur mixed in a single right hand side.

BES with such right-hand sides occur *e.g.* in the context of process equivalence checking problems [CPPW07], and more complex model checking problems. Recall the model checking example from Chapter 2.

Example 6.2.5. The unreliable channel can read messages from the environment, and send or lose these next. 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.



We use the formula that expresses 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. The problem is formalised as follows:

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

Using translation of Mader [Mad97], the BES given below is obtained. The solution to X_{s_i} answers whether s_i satisfies formula ϕ.

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

As illustrated by this example, BES with mixed occurrences of $\wedge$ and $\vee$ occur in realistic model checking examples. To eliminate the translation to SRF we show that there indeed is a progress measure on closed BESs with arbitrary right hand sides. We use the following property of the translation of a BES to SRF.

Lemma 6.2.6. Given a BES $\mathcal{E}$ in recursive form, all cycles in $\mathcal{E}$ are ν-dominated if and only if all cycles in the corresponding BES $\mathcal{E}'$ in SRF are ν-dominated.

Proof

$\Rightarrow$) Assume we have a BES $\mathcal{E}$ in which all cycles are ν-dominated. We show that all cycles in the corresponding BES $\mathcal{E}'$ in SRF (obtained by applying the translation from Section 3.2) are ν-dominated.

We show that the translation does not introduce μ-dominated cycles in $\mathcal{E}'$. We assume that there is a μ-dominated cycle in $\mathcal{E}'$. All edges introduced by the translation to SRF are of the form $X \to X_1$, $X_i \to X_{i+1}$ or $X_n \to Y$ in $\mathcal{E}'$. In $\mathcal{E}$ there were edges $X \to Y$. Furthermore we know that each X_i has got exactly one incoming edge. Hence, if there is a cycle $Z \to \cdots \to X \to X_1 \to \cdots \to X_n \to Y \to \cdots \to Z$, then there originally was a cycle $Z \to \cdots \to X \to Y \to \cdots \to Z$. Because $\mathsf{rank}(X_i) = \mathsf{rank}(X)$ for all i, the original cycle must also be μ-dominated, which is a contradiction to the assumption that $\mathcal{E}$ does not contain μ-dominated cycles.

$\Leftarrow$) Assume we have a BES $\mathcal{E}'$ in simple form, which is the result of normalizing a BES, in which all cycles are ν-dominated. We need to show that in any BES $\mathcal{E}$ of which $\mathcal{E}'$ is the simple form equivalent all cycles are ν-dominated. The proof of this direction follows a similar line of reasoning to the other case. Assume there is a μ-dominated cycle in $\mathcal{E}$, and show that than there also must have been a μ-dominated cycle in $\mathcal{E}'$, deriving a contradiction. □

Theorem 6.2.7. There is a progress measure for a BES $\mathcal{E}$ if and only if all cycles in dependency graph $\mathcal{G}_\mathcal{E}$ of $\mathcal{E}$ are ν-dominated.

Proof Follows directly from Lemma 6.2.6 and Theorem 6.2.2. □

We extend the definitions of $\mathsf{Prog}(\_,\_,\_)$ and $\mathsf{Lift}(\_,\_)(\_)$ from Section 6.2 such that they can be applied to BESs in RF. Given $\varrho{:}\mathcal{X} \to M^\top$ and propositional formula f, then $\mathsf{Prog}(\varrho, X, f)$ is the least $m \in M^\top$ such that

$$\begin{cases} m \geqslant_{\mathsf{rank}(X)} \varphi(\varrho, f) & \text{if } \mathsf{rank}(X) \text{ is even} \\ m >_{\mathsf{rank}(X)} \varphi(\varrho, f) & \text{if } \mathsf{rank}(X) \text{ is odd} \end{cases}$$

where $\varphi(\varrho, f)$ is defined inductively as follows:

$$\varphi(\varrho, f) = \begin{cases} \varrho(Y) & \text{if } f = Y \\ \min\{\varphi(\varrho, f_i)\} & \text{if } f = \bigvee_i f_i \\ \max\{\varphi(\varrho, f_i)\} & \text{if } f = \bigwedge_i f_i \end{cases}$$

A function $\varrho{:}\mathcal{X} \to M^\top$ is a BES progress measure if and only if for all X in $\mathcal{X}$:

$$\begin{cases} \varrho(X) \geqslant_{\mathsf{rank}(X)} \mathsf{Prog}(\varrho, X, Y) & \text{if } \mathsf{rhs}(X, \mathcal{E}) = Y \\ \exists_{f_i}\, \varrho(X) \geqslant_{\mathsf{rank}(X)} \mathsf{Prog}(\varrho, X, f_i) & \text{if } \mathsf{rhs}(X, \mathcal{E}) = \bigvee_i f_i \\ \forall_{f_i}\, \varrho(X) \geqslant_{\mathsf{rank}(X)} \mathsf{Prog}(\varrho, X, f_i) & \text{if } \mathsf{rhs}(X, \mathcal{E}) = \bigwedge_i f_i \end{cases}$$

By $||\varrho||$ we denote the set $\{X \in \mathcal{X} | \varrho(X) \neq \top\}$.

The family of lifting functions $\mathsf{Lift}(\_, X)(\_)$ is extended to the general setting as follows:

$$\mathsf{Lift}(\varrho, X)(Y) = \begin{cases} \varrho(Y) & \text{if } X \neq Y \\ \mathsf{Prog}(\varrho, X, Z) & \text{if } X = Y \text{ and } \mathsf{rhs}(X, \mathcal{E}) = Z \\ \min\{\mathsf{Prog}(\varrho, X, f_i)\} & \text{if } X = Y \text{ and } \mathsf{rhs}(X, \mathcal{E}) = \bigvee_i f_i \\ \max\{\mathsf{Prog}(\varrho, X, f_i)\} & \text{if } X = Y \text{ and } \mathsf{rhs}(X, \mathcal{E}) = \bigwedge_i f_i \end{cases}$$

Observe that in case the BES is in SRF, the definitions given in this section are equivalent to the definitions from Section 6.2. We show this in the following example.

Example 6.2.8. Consider an equation $\sigma X = Y$, and a progress measure ϱ, then $\mathsf{Lift}(\varrho, X)(X)$ equals $\mathsf{Prog}(\varrho, X, Y)$, which is in turn the least m such that $m \geqslant_{\mathsf{rank}(X)} \varphi(\varrho, Y)$, which is the least m such that $m \geqslant_{\mathsf{rank}(X)} \varrho(Y)$ according to the definition of φ. Similar observations hold for conjunctive and disjunctive equations, using the fact that no mixing of conjuncts and disjuncts occurs in a single equation.

It is not at all clear whether a witness can be computed directly from the progress measure in its general form as provided in this section. Corresponding results are therefore absent. Note that this does not form a restriction, as we are generally interested in the solution of the BES, and not its witness. It does, however, form an inconvenience, as the witness may be used by the user in convincing himself of the correctness of the solution.

We show the correctness of the generalised algorithm by showing that the progress measure computed for a BES in RF is the same as the progress measure computed for the BES obtained through a naive translation to SRF.

For the sake of argument we consider the following translation to SRF. Let $\mathcal{E}$ be a BES in RF. We define the augmented BES $\mathcal{E}_{aug}$ as follows. For each equation $\sigma X = f \vee g$, where g is a subformula containing $\wedge$ as top-level symbol we introduce an equation $\nu X_g = g$ at the end of $\mathcal{E}$. Note that X_g is fresh. This transformation is applied recursively to the newly introduced equations. A similar transformation is done for f, if f is a subformula containing $\wedge$ as top-level symbol. The translation for equations $\sigma X = f \wedge g$ with g a subformula with $\vee$ as top level symbol is similar. The BES $\mathcal{E}_{aug}$ hence is the BES $\mathcal{E}$ with subformulae with non-trivial alternations appended in a large block of greatest fixpoint equations. Note that $\mathcal{E}$ and $\mathcal{E}_{aug}$ have the same solutions for $X \in \mathsf{bnd}(\mathcal{E})$, *i.e.* for all $X \in \mathsf{bnd}(\mathcal{E})$ it holds that $[\![\mathcal{E}]\!](X) = [\![\mathcal{E}_{aug}]\!](X)$. Furthermore $[\![\mathcal{E}_{aug}]\!] \equiv [\![\mathcal{E}_{SRF}]\!]$. Now define the corresponding BES in SRF $\mathcal{E}_{SRF}$ to be $\mathcal{E}_{aug}$ where the occurrences of formulae g for which a corresponding equation $\nu X_g = g$ was introduced are replaced by X_g. We show that $\mathcal{E}_{aug}$ and $\mathcal{E}_{SRF}$ have the same progress measure in the fixed point.

Lemma 6.2.9. Let $\mathcal{E}_{aug}$ be an augmented BES, let $\mathcal{E}_{SRF}$ be the corresponding BES in SRF. Consider progress measures ϱ', which is a fixpoint of Lift on $\mathcal{E}_{aug}$, and ϱ, which is a fixpoint of Lift on $\mathcal{E}_{SRF}$. So, ϱ' and ϱ are the solutions of applying *GeneralBESProgressMeasures* on the corresponding BESs. It holds that $\varrho = \varrho'$.

Proof We prove that $\varrho = \varrho'$ by showing that backwards substitution of augmenting equations in $\mathcal{E}_{SRF}$ preserves ϱ. Let a *substitution step* be defined as taking the last equation $\sigma X = f$ in the BES, and applying the substitution $X := f$ to the last occurrence of X in the right hand side in the BES. If X does not occur in any right hand side we consider the equation preceding X as substitution.

Let $\mathcal{E}^i_{SRF}$ be $\mathcal{E}_{SRF}$ with i substitution steps applied, similarly, ϱ^i is the progress measure that results of applying the algorithm on $\mathcal{E}^i_{SRF}$. We prove preservation of ϱ by backwards substitution with induction on the number of substitution steps n, *i.e.* we show that for all values of n it holds that $\varrho^n = \varrho$. The base case, with 0 substitution steps is trivial, as $\mathcal{E}^0_{SRF} = \mathcal{E}_{SRF}$, hence $\varrho^0 = \varrho$. Now we assume induction hypothesis $\varrho^n = \varrho$. We need to show that $\varrho^{n+1} = \varrho$. We show this for an equation with $\wedge$ as top-level symbol. The case with $\vee$ as top-level symbol is analogous. Without loss of generality assume that $\mathcal{E}^n_{SRF}$ is of the form

$$\mathcal{E}_0(\sigma X = f \wedge X_{g \vee h})\mathcal{E}_1(\nu X_{g \vee h} = g \vee h)\mathcal{E}_2$$

A single application of the substitution $X_{g \vee h} := g \vee h$ leads to the following BES $\mathcal{E}^{n+1}_{SRF}$

$$\mathcal{E}_0(\sigma X = f \wedge (g \vee h))\mathcal{E}_1(\nu X_{g \vee h} = g \vee h)\mathcal{E}_2$$

Observe that this substitution can only change the progress measure for X. The lemma therefore follows if we can show that $\varrho^{n+1}(X) = \varrho^n(X)$. That this equivalence holds is shown in the following derivation.

$$\begin{aligned}
& \varrho^{n+1}(X) \\
= \quad & \{\ \varrho^{n+1} \text{ is a fixed point for } \mathsf{Lift}\ \} \\
& \mathsf{Lift}(\varrho^{n+1}, X)(X) \\
= \quad & \{\text{Definition of } \mathsf{Lift}\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \mathsf{Prog}(\varrho^{n+1}, X, g \vee h)) \\
= \quad & \{\text{Definition of } \mathsf{Prog}\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \varphi(\varrho^{n+1}, g \vee h)) \\
= \quad & \{\text{Definition of } \varphi\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \min(\varphi(\varrho^{n+1}, g), \varphi(\varrho^{n+1}, h))) \\
= \quad & \{\text{Definition of } \varphi\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \min(\mathsf{Prog}(\varrho^{n+1}, X_{g \vee h}, g), \mathsf{Prog}(\varrho^{n+1}, X_{g \vee h}, h))) \\
= \quad & \{\text{Definition of } \mathsf{Lift}\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \mathsf{Lift}(\varrho^{n+1}, X_{g \vee h})(X_{g \vee h})) \\
= \quad & \{\text{Definition of } \mathsf{Prog}\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \mathsf{Prog}(\varrho^{n+1}, X_{g \vee h}, g \vee h)) \\
= \quad & \{\varrho^{n+1} \text{ is a fixed point}\} \\
& \max(\mathsf{Prog}(\varrho^{n+1}, X, f), \mathsf{Prog}(\varrho^{n+1}, X, g \vee h)) \\
= \quad & \{\varrho^{n+1}(X_{g \vee h}) = \varrho^n(X_{g \vee h})\} \\
& \max(\mathsf{Prog}(\varrho^{n}, X, f), \mathsf{Prog}(\varrho^{n}, X, g \vee h)) \\
= \quad & \{\text{Definition of } \mathsf{Lift}\} \\
& \mathsf{Lift}(\varrho^{n}, X)(X) \\
= \quad & \{\ \varrho^{n} \text{ is a fixed point for } \mathsf{Lift}\ \} \\
& \varrho^{n}(X)
\end{aligned}$$

As noted, the proof for equations with $\vee$ as top-level symbol is analogous, and the lemma follows. □

Observe that augmenting a BES does not alter the progress measure for already existing equations (as the newly added equations are not used in the rest of the BES). Furthermore, given that m equations are introduced in augmenting, $\mathcal{E}^m_{SRF} = \mathcal{E}_{aug}$, and the proof shows that therefore $\varrho^m = \varrho'$. We may thus apply the generalised version of small progress measures to a BES in RF.

The following lemma shows that the solution of a BES in RF can be characterised using the progress measure.

Lemma 6.2.10. Let $\mathcal{E}$ be a BES in RF. There is a BES progress measure $\varrho{:}\mathsf{bnd}(\mathcal{E}) \to M^\top$ such that $||\varrho||$ is the set of propositional variables that are true in $\mathcal{E}$.

Proof Follows directly from Lemma 6.2.9 and Theorem 6.2.2. □

Using these definitions, we can apply the same algorithm as in Section 6.2.

Algorithm 3 GeneralBESProgressMeasureLifting

$\mu \leftarrow \lambda X \in \mathsf{bnd}(\mathcal{E}).\overline{0}$
while $\mu \sqsubset \lambda Y \in \mathsf{bnd}(\mathcal{E}).\mathsf{Lift}(\mu, X)(Y)$ for some $X \in \mathsf{bnd}(\mathcal{E})$ **do**
  $\mu \leftarrow \lambda Y \in \mathsf{bnd}(\mathcal{E}).\mathsf{Lift}(\mu, X)(Y)$
end while

Observe that in general the algorithm in this section runs on BESs with a smaller number of equations, and the sum of the sizes of the right hand sides will be smaller. Also the time required for each step in the algorithm remains the same. The algorithmic complexity of the algorithm is the same as the algorithm for BESs in SRF, but in general the constants will be smaller, and the algorithm can be expected to perform better.

Theorem 6.2.11. Given a BES $\mathcal{E}$, with n equations, and cumulative size e of its right hand sides, Algorithm 3 computes the solution of $\mathcal{E}$ in time $O(\mathsf{ah}(\mathcal{E}) \cdot e \cdot \frac{n}{\mathsf{ah}(\mathcal{E})}^{\mathsf{ah}(\mathcal{E})/2})$.

6.3 Summary

We have demonstrated that the small progress measures algorithm can be applied to BESs in SRF straightforwardly. In addition we have shown that the algorithm can be generalised such that it can be applied to BESs in RF. This results in an algorithm for solving BESs that does not require preprocessing involving a linear blow-up of the BES. Note that in itself this is not a novelty–BES algorithms like Gauß elimination can also be applied to bes with arbitrary right hand sides–but it does indicate that the performance of parity game algorithms might be improved when we apply them to BESs. Furthermore the generalisation of small progress measures indicates that it may be possible to generalise other parity game algorithms in the BES framework. The most interesting algorithm to consider in that respect is bigstep, which was demonstrated to be efficient in the previous chapter, and uses small progress measure as subroutine.

Chapter 7

Conclusions

We have investigated algorithms and techniques to improve the performance of solving parity games. It turns out that performing decomposition into strongly connected components prior to solving, as well as solving special cases with specialised algorithms give significant improvements. We have also shown that in practice the recursive and bigstep algorithms show the best performance. It was demonstrated that there is room for generalisation of the algorithms when applied to Boolean Equation Systems by showing a generalisation of the small progress measures algorithm.

We have presented the connection between Boolean Equation Systems and parity games in detail. Investigating the equivalence between both frameworks gives rise to a more thorough understanding of BESs. As a result of this we have presented some new manipulations on BESs, as well as some meta-techniques that can be used in all solvers that work on BESs in SRF.

In addition we have compared a large number of algorithms for solving parity games, using concrete model checking examples as input. Furthermore the effect of the meta-techniques for solving parity games is investigated, as well as the effect of an increasing number of priorities on the running time of the algorithms.

It was shown that decomposition of a parity game into strongly connected components and the solving of special cases using an efficient tailor-made algorithm vastly improve the running time of all algorithms. The optimisations also decrease the relative differences between the various algorithms, hence diminishing the dramatic difference between algorithms as it occurs in the literature.

The effect seen when increasing the number of priorities in the parity games on the running time coincides with the expectation based on the theoretic worst case running times. Our results show that indeed the algorithms that depend on the number of priorities show this dependency in practice, but only as long as optimisations are switched off.

The literature provides hardly any experiments concerning the running times of parity game solving algorithms. Furthermore all experiments that are available treat pathologic examples instead of concrete model checking examples. Therefore this work may serve as a basis for further development of the area of parity game solving, and even model checking in general. The most important and somewhat unsettling conclusion of the experiments is that even though a tremendous amount of work has been carried out to find more efficient algorithms for solving parity games during the past decades, in general they are all beaten by one of the most straightforward algorithms based on an elegant determinacy proof. On the other hand it is shown by the results concerning the bigstep algorithm that combining different algorithms may lead to good results as well.

As we have looked at these parity game algorithms with an application to Boolean Equation Systems in the back of our minds, we have shown that parity game algorithms can straightforwardly be applied to BESs in standard recursive form by transforming the small progress measures algorithm to this framework. Additionally we have demonstrated how this algorithm can be generalised to be applicable to BESs with arbitrary right hand sides.

Future work The choice for concentrating on the small progress measures algorithm for showing the feasibility of applying parity game algorithm to BESs, and generalisation of the resulting algorithm was made, based on its appealing theoretic worst-case running times. Although the algorithm was shown not to be the most effective in practice, it is also part of the bigstep algorithm which was successful, hence the generalisation may serve as a first step towards generalising the full bigstep algorithm.

We think it is an interesting exercise to implement the most efficient algorithms, the recursive algorithm and the bigstep algorithm, directly in the BES framework, hence to eliminate the transformation from BES to parity game. That also paves the way for investigating generalisation of these algorithms to work on arbitrary BESs, similar to what we have demonstrated with small progress measures. Note that a lot of verification problems give rise to systems in standard form. However, the class of BESs generated for *e.g.* branching bisimilarity is typically not in standard form, *i.e.* conjuncts and disjuncts occur mixed in a single right hand side. Therefore, generalising the algorithms in the framework of BESs will probably pose an improvement for this class of systems.

Implementations in the BES framework of both the original algorithms, and possible generalised versions could be compared to the experiments we have presented. This should at least make it possible to compare the performance of the PGSolver toolset with other implementations. Furthermore the effect of the generalisation of the algorithms on the running times could be shown.

Besides investigating efficient algorithms for solving BESs and parity games, reduction techniques for BESs based on equivalence (weaker than strong bisimulation and oblivious bisimulation) should be further investigated. Furthermore similar techniques on the symbolic level of Parameterized Boolean Equation Systems (PBESs) should be investigated.

Bibliography

[BC04] Y. Bertot and P. Castéran. *Interactive theorem proving and program development; Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EACTS Series. 2004.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CPPW07] T. Chen, B. Ploeger, J.v.d. Pol, and T.A.C. Willemse. Equivalence checking for infinite systems using parameterized boolean equation systems. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR 2007)*, volume 4703 of *LNCS*, pages 120–135. Springer, 2007.

[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*, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society.

[EL86] E. A. Emerson and C.L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of LICS 1986*, pages 267–278. IEEE Computer Society, 1986.

[FL09] O. Friedmann and M. Lange. The PGSolver collection of parity game solvers. Technical report, Institut für Informatik, Ludwig-Maximilians-Universität München, Germany, 2009.

[Fri09] O. Friedmann. A super-polynomial lower bound for the parity game strategy improvement algorithm as we know it. 2009. To appear in LICS2009; available from the author's website `http://www.tcs.informatik.uni-muenchen.de/~friedman/`.

[GA94] M.J.C. Gordon and A.M.Pitts. The HOL logic and system. In J. Bowen, editor, *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*, chapter 3, pages 49–70. Elsevier Science B.V., 1994.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 60–65, New York, NY, USA, 1982. ACM.

[GK05] J.F. Groote and M. Keinänen. A sub-quadratic algorithm for conjunctive and disjunctive boolean equation systems. In *Theoretical Aspects of Computing – ICTAC 2005*, volume 3722 of *Lecture Notes in Computer Science*, pages 532–545, Berlin/Heidelberg, 2005. Springer.

[GLMS07] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. Cadp2006: A toolbox for the construction and analysis of distributed processes. In *Proceedings of the 19th International Conference on Computer Aided Verification CAV2007*, 2007.

[GM99] J.F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In A.M. Haeberer, editor, *AMAST'98*, volume 1548 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999.

[GMWU07] J.F. Groote, A. Mathijssen, M. v. Weerdenburg, and Y.S. Usenko. The formal specification language mcrl2. In *Proceedings of Methods for Modelling Software Systems*, volume 06351 of *Dagstuhl Seminar Proceedings*, 2007.

[GP08] J.F. Groote and B. Ploeger. Switching graphs. *Electron. Notes Theor. Comput. Sci.*, 223:119–135, 2008.

[GW05] J.F. Groote and T.A.C. Willemse. Parameterised boolean equation systems. *Theoretical Computer Science*, 343(3):332–369, 2005. Formal Methods for Components and Objects.

[JPZ06] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 117–123, New York, NY, USA, 2006. ACM.

[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*, pages 290–301, London, UK, 2000. Springer-Verlag.

[Kei06] M.K. Keinänen. *Solving Boolean Equation Systems*. PhD thesis, Helsinki University of Technology, Department of Computer Science and Engineering, April 2006.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[KW09] J. Keiren and T.A.C. Willemse. Bisimulation minimisations for boolean equation systems, 2009. Submitted for publication. To appear as technical report.

[Lan05] M. Lange. Solving parity games by a reduction to sat. In *In Proc. of the Workshop on Games in Design and Verification, GDV05*, 2005.

[Mad97] A. Mader. *Verification of Modal Properties Using Boolean Equation Systems*. PhD thesis, Technische Universität München, 1997.

[Mar75] D.A. Martin. Borel determinacy. *The Annals of Mathematics*, 102(2):363–371, 1975.

[Mat97] R. Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, INRIA Rhne-Alpes, 1997.

[Mat03] R. Mateescu. A generic on-the-fly solver for alternation-free boolean equation systems. In *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2003.

[McN93] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[ORS92] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[Pau86] L.C. Paulson. Natural deduction as higher-order resolution. *J. Log. Program.*, 3(3):237–258, 1986.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, 11 1977.

[PT87] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *Siam J. Comput.*, 16(6), 1987.

[PW08] J. v.d. Pol and Michael Weber. A multi-core solver for parity games. In I. Cerna and G. Lüttgen, editors, *Proceedings of PDMC 2008*, volume 220, pages 19–34. Elsevier Science Publishers B. V., 2008.

[RW09] M.A. Reniers and T.A.C. Willemse. Analysis of boolean equation systems through structure graphs, 2009. Submitted for publication.

[Sch07] S. Schewe. Solving parity games in big steps. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, volume 4855, pages 449–460, Berlin/Heidelberg, 2007. Springer.

[Sch08] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Computer Science Logic*, volume 5213, pages 369–384, Berlin/Heidelberg, 2008. Springer.

[SS98] P. Stevens and C. Stirling. Practical model-checking using games. In *TACAS '98: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 85–101, London, UK, 1998. Springer-Verlag.

[Sti95] C. Stirling. Local model checking games. In *CONCUR '95: Proceedings of the 6th International Conference on Concurrency Theory*, pages 1–11, London, UK, 1995. Springer-Verlag.

[Sti96] C. Stirling. Model checking and other games. Notes for Mathfit Workshop on Finite Model Theory, University of Wales Swansea, 1996.

[SV00] D. Schmitz and J. Vöge. Implementation of a strategy improvement algorithm for finite-state parity games. In *CIAA*, pages 263–271, 2000.

[Use02] Y.S. Usenko. *Linearisation in μCRL*. PhD thesis, Eindhoven University of Technology, December 2002.

[vDPW] A. van Dam, B. Ploeger, and T.A.C. Willemse. Instantiation for parameterised boolean equation systems.

[VJ00] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV*, pages 202–215, 2000.

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998.

Appendix A

Experimental results for optimisation techniques

In this chapter we present the results of our experiments showing the effects of various optimisation techniques in solving parity games, applied to concrete model checking examples. In Section A.1 we show the effects of decomposition into strongly connected components. In Section A.2 we show the influence of solving special games, assuming that SCC decomposition is always enabled.

The large set of numeric data that results from this experiment is hard to interpret in its raw form. As we are looking for the effects of optimisations on the running time of the experiments, we have created scatter plots where on the x-axis the experiment number (determined by the optimisations were enabled) and on the y-axis the running time have been set out. Note that runs with the lowest running time, and hence the best performance are found in the top of the graphs. Each graph contains either 128 or 64 points, each of which identifies a run of PGSolver on a specific problem instance (consisting of a single specification with a single modal formula) with the same algorithm, but with different combinations of the optimisations enabled (*i.e.* 2^7 combinations of the 7 optimisations that we have described). In the cases where only 64 measurements are depicted decomposition into strongly connected components was always enabled. Clusters in these graphs are courtesy of certain combinations of optimisations.

We consider an optimisation to be relevant if all runs where the optimisation is disabled show a running time that is significantly longer than the fastest running time.

In the figures, all measurements that have been carried out are marked with $+$, in addition, experiments where a certain optimisation was disabled are marked with X. Exactly which optimisation was disabled is marked in the legend of each graphs.

Note that a value of -1 in the graphs means that PGSolver did not terminate within 30 minutes, and -2 indicates that PGSolver terminated with a memory error.

A.1 SCC Decomposition

In this section we present an overview of the measurements that establish the usefulness of SCC decomposition in the process of solving parity games, using various algorithms. In each of the plots the cases where SCC decomposition is disabled have been marked, showing that overall it is advisable to enable SCC decomposition.

This section has been split up according to the modal properties that have been tested.

A.1.1 Deadlock freedom

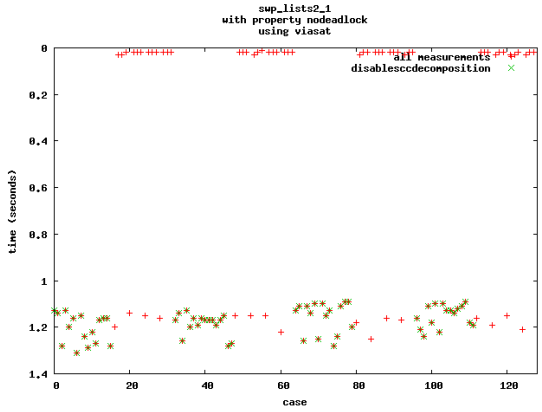


Figure A.1: Viasat

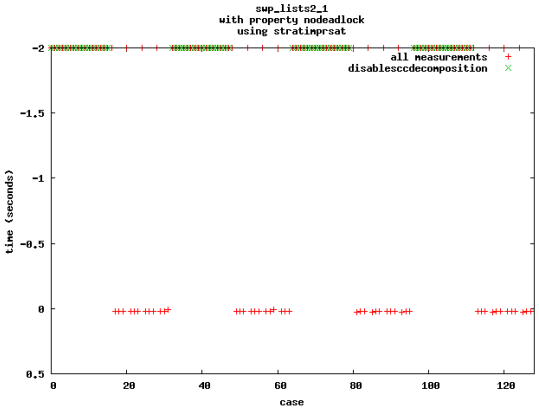


Figure A.2: Stratimprsat

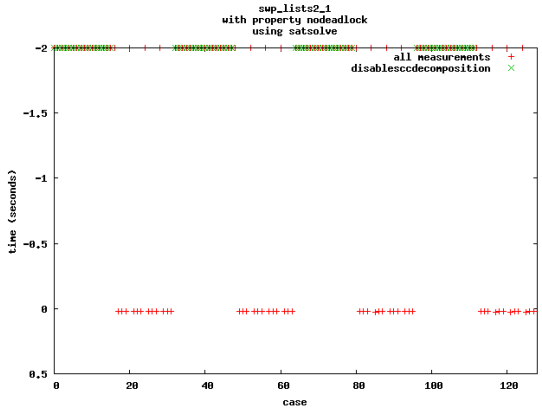


Figure A.3: Satsolve

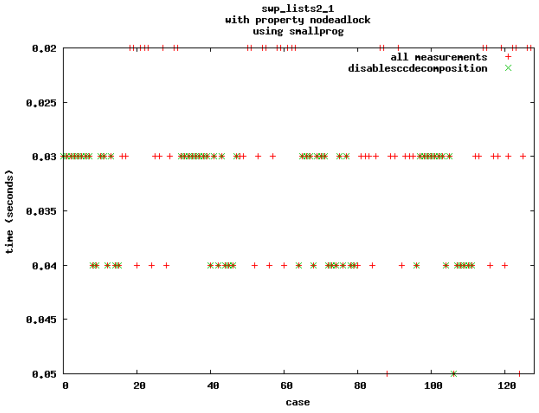


Figure A.4: Small progress measures

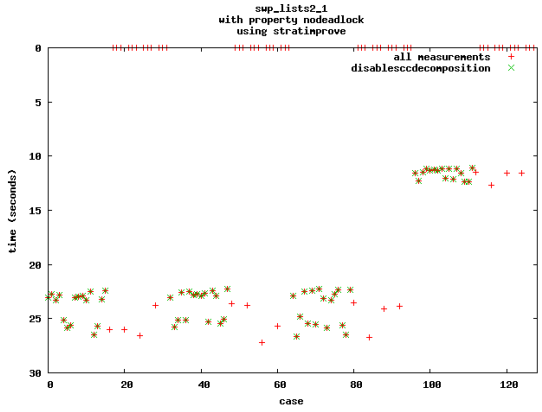


Figure A.5: Strategy improvement

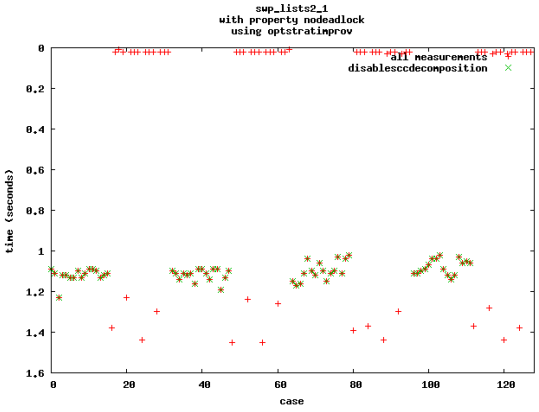


Figure A.6: Optimal strategy improvement

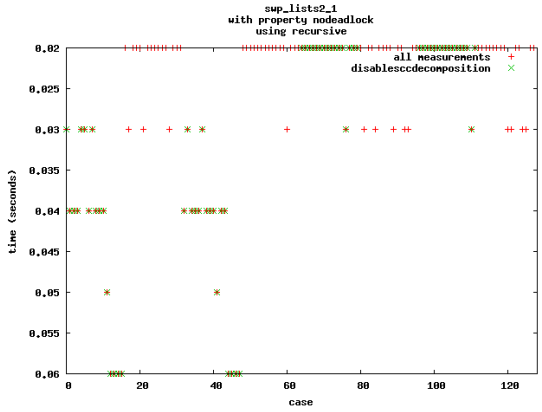


Figure A.7: Recursive

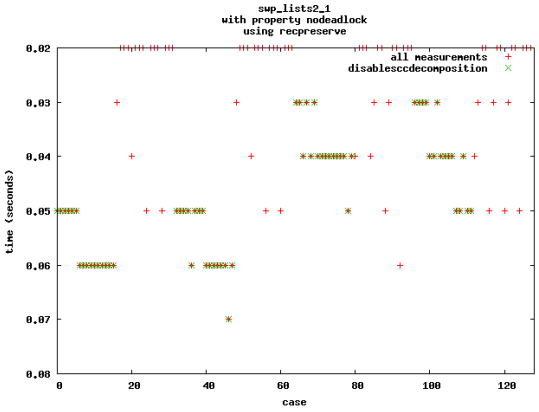


Figure A.8: Recursive preservation

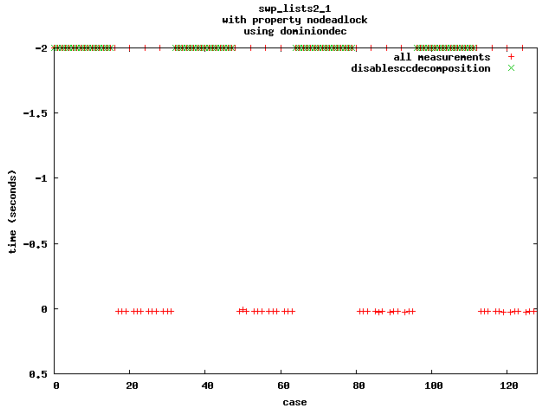


Figure A.9: Dominion decomposition

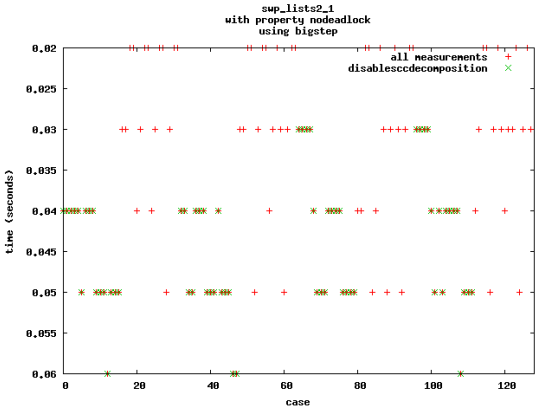


Figure A.10: Bigstep

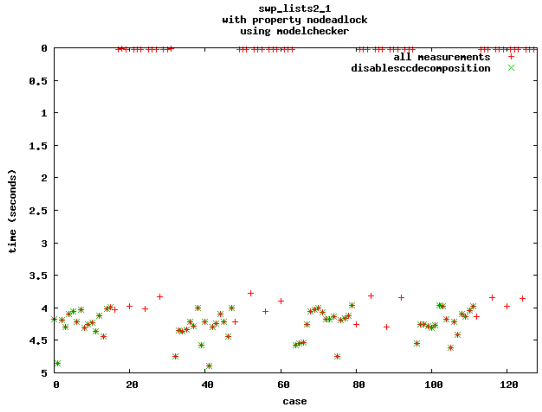


Figure A.11: Model checker

A.1.2 Livelock freedom

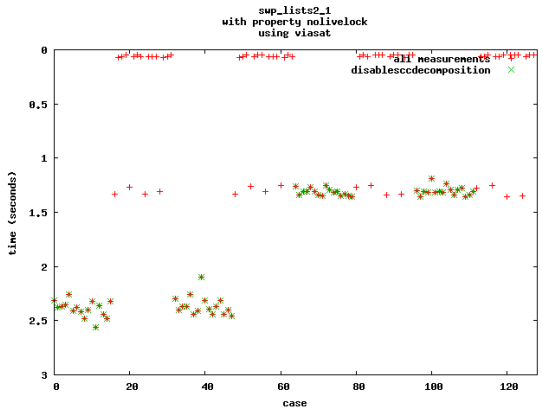


Figure A.12: Viasat

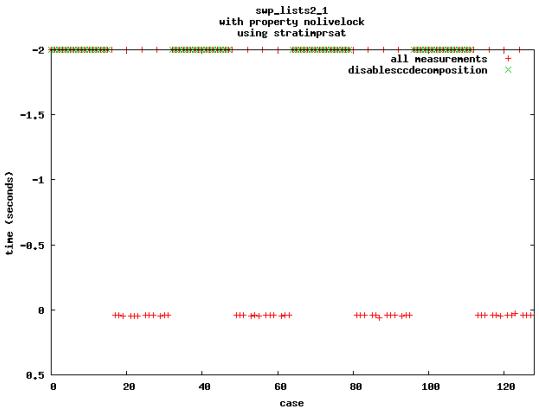


Figure A.13: Stratimprsat

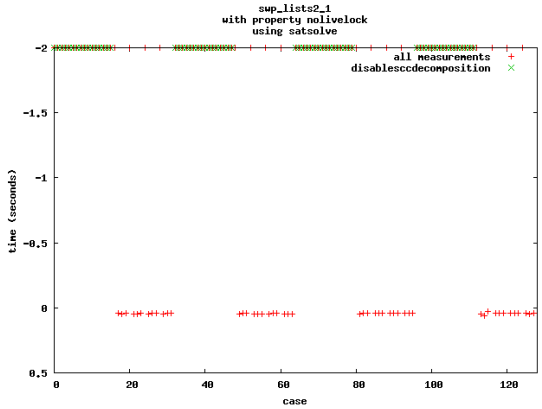


Figure A.14: Satsolve

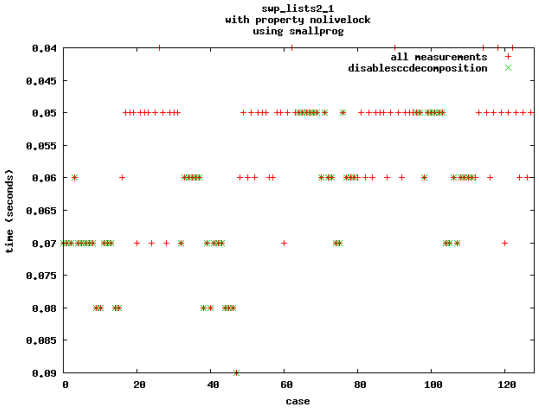


Figure A.15: Small progress measures

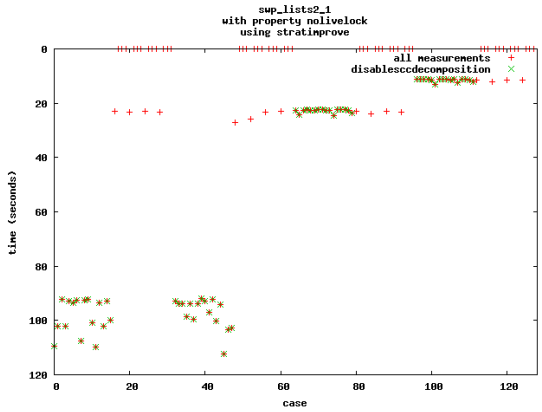


Figure A.16: Strategy improvement

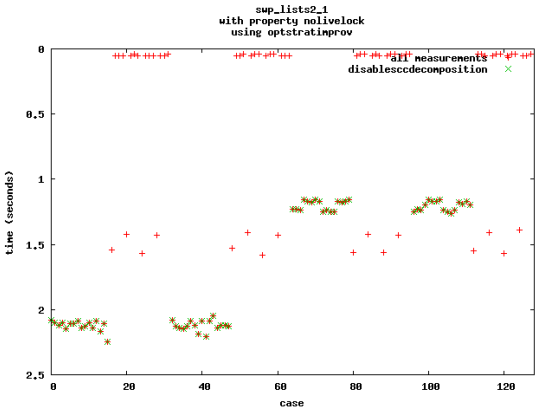


Figure A.17: Optimal strategy improvement

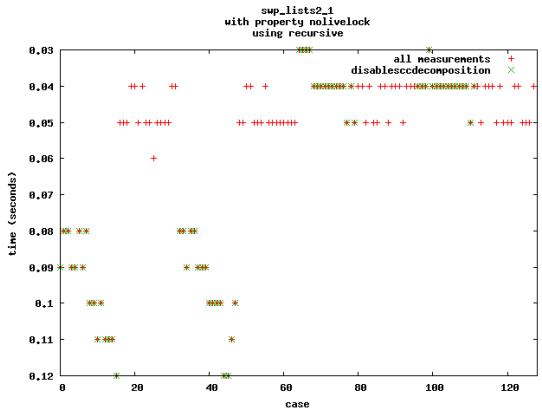


Figure A.18: Recursive

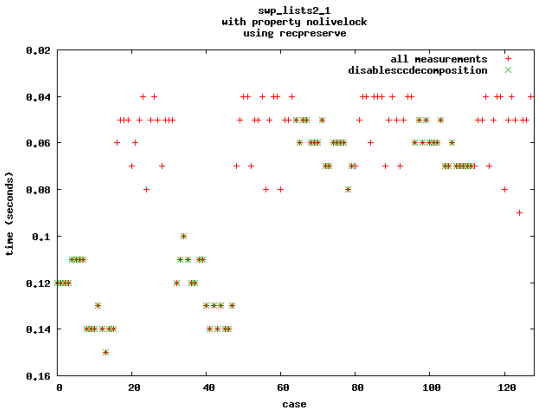


Figure A.19: Recursive preservation

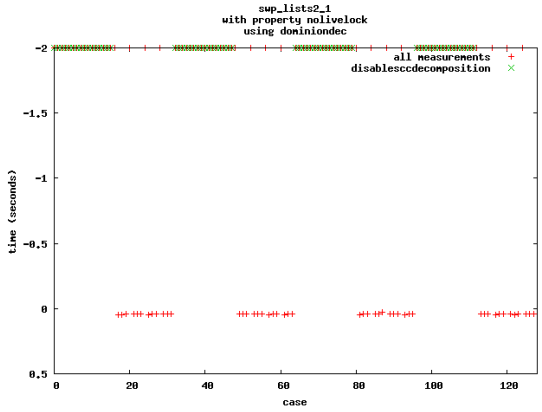


Figure A.20: Dominion decomposition

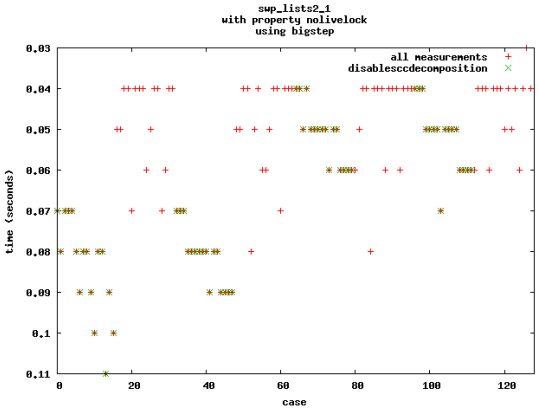


Figure A.21: Bigstep

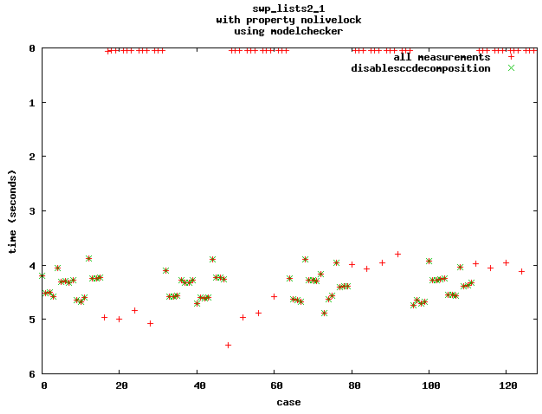


Figure A.22: Model checker

A.1.3 Infinitely often receive

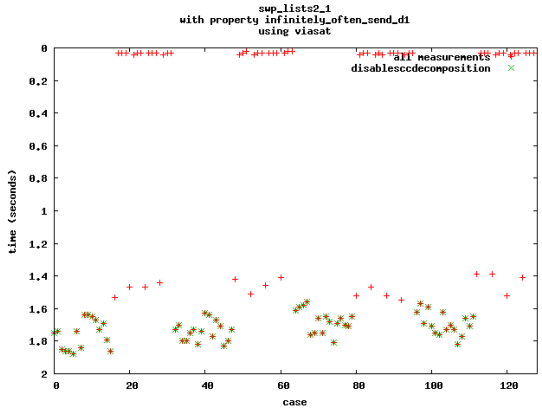


Figure A.23: Viasat

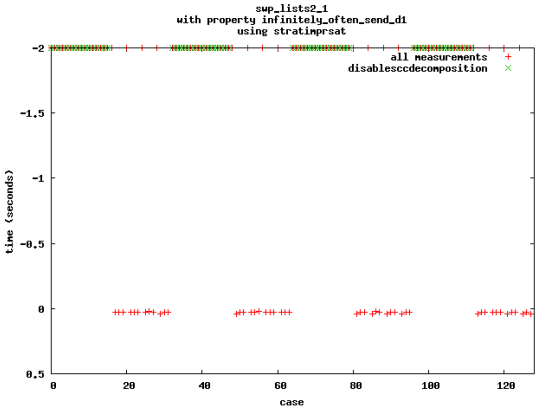


Figure A.24: Stratimprsat

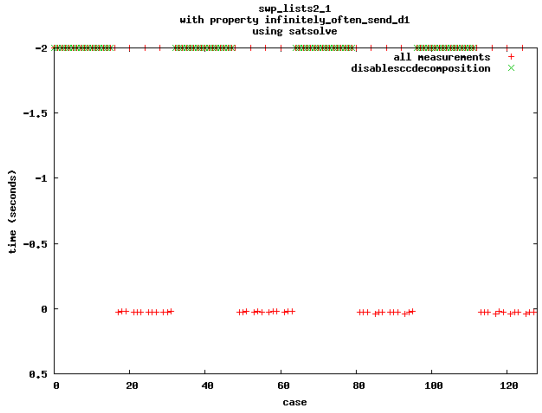


Figure A.25: Satsolve

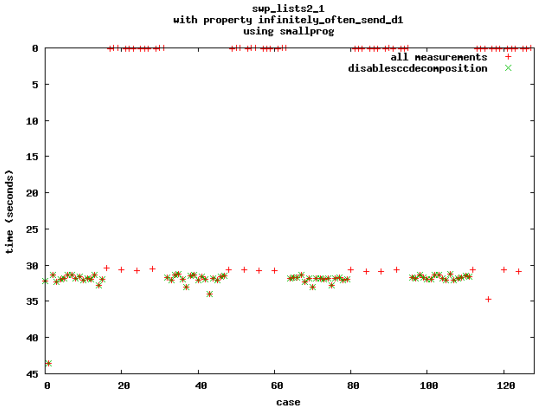


Figure A.26: Small progress measures

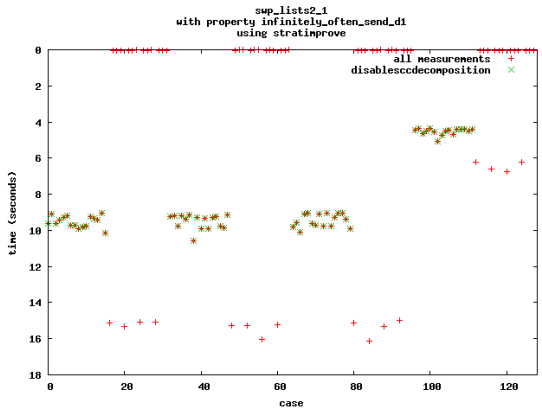


Figure A.27: Strategy improvement

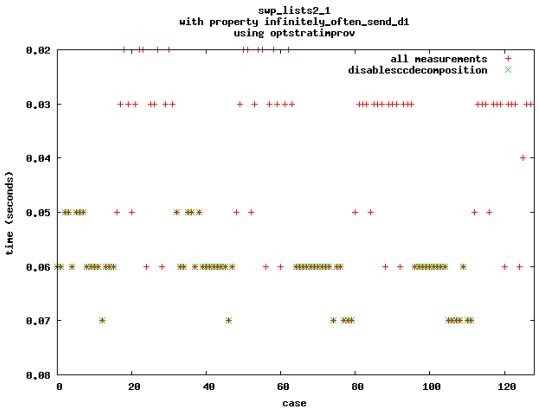


Figure A.28: Optimal strategy improvement

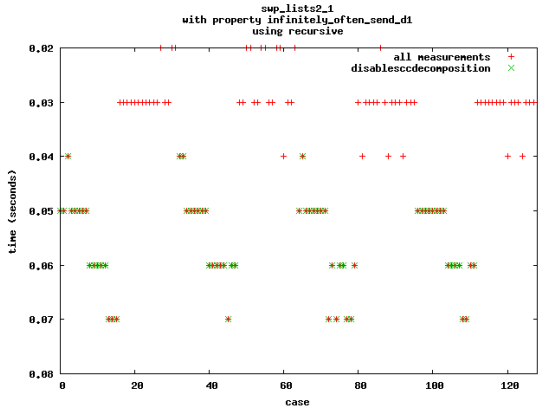


Figure A.29: Recursive

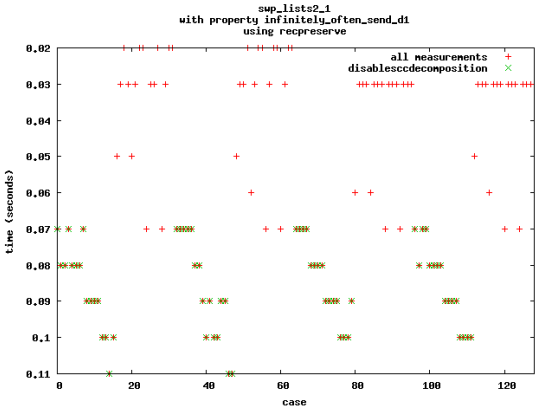


Figure A.30: Recursive preservation

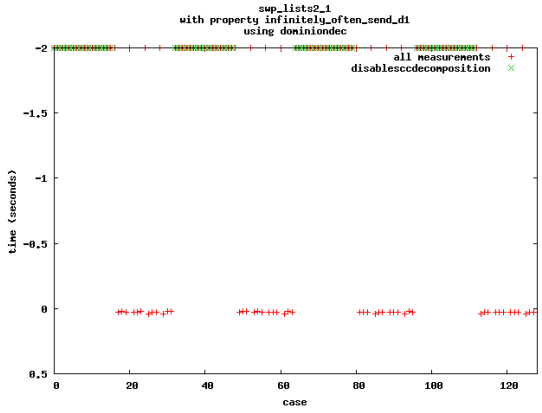


Figure A.31: Dominion decomposition

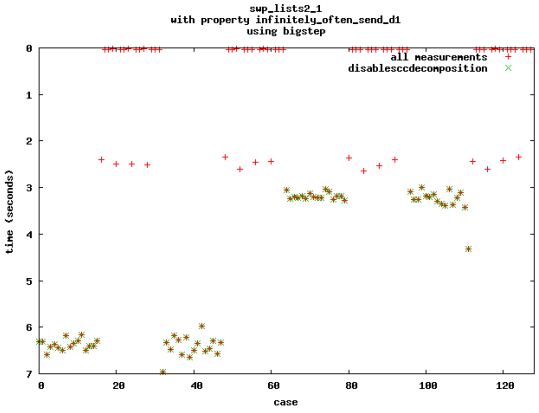


Figure A.32: Bigstep

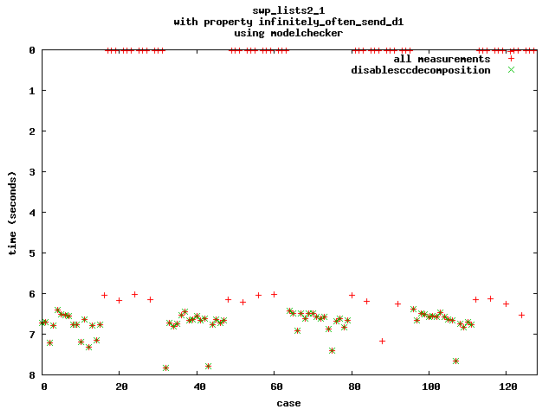


Figure A.33: Model checker

A.1.4 Infinitely often enabled, then infinitely often taken

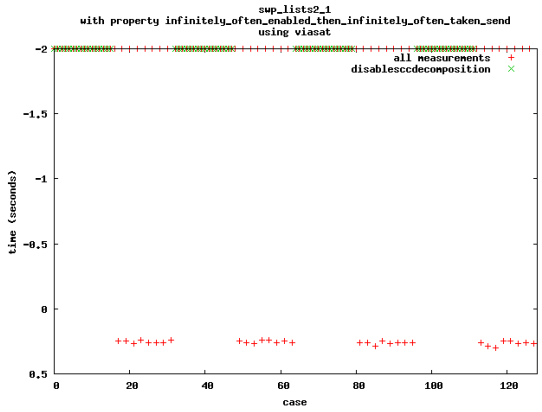


Figure A.34: Viasat

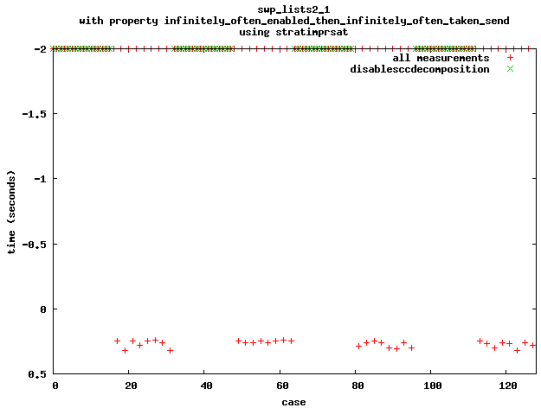


Figure A.35: Stratimprsat

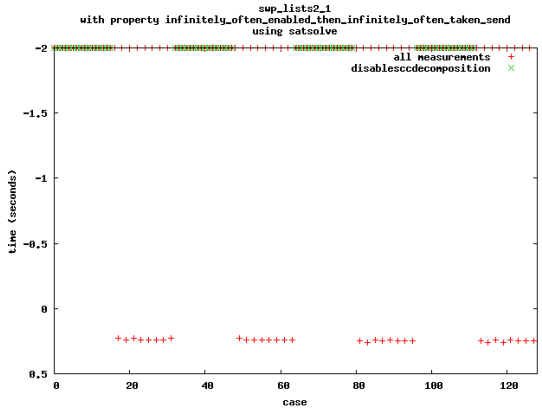


Figure A.36: Satsolve

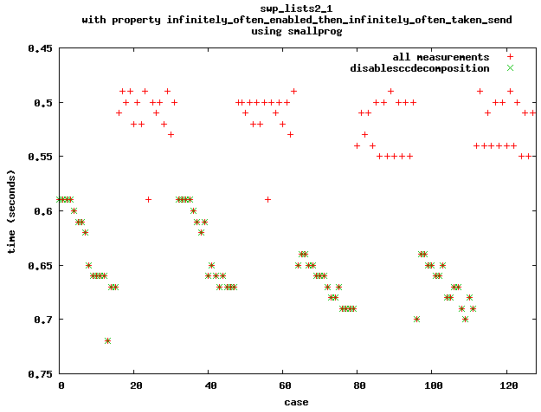


Figure A.37: Small progress measures

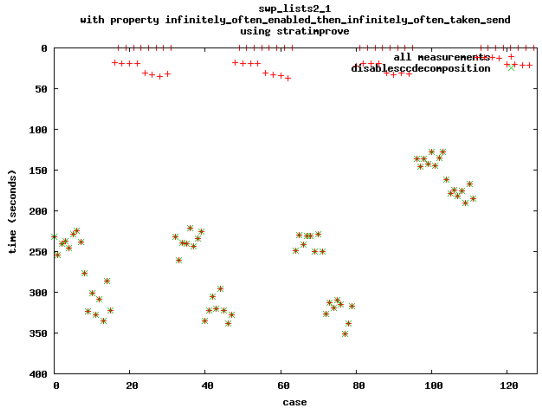


Figure A.38: Strategy improvement

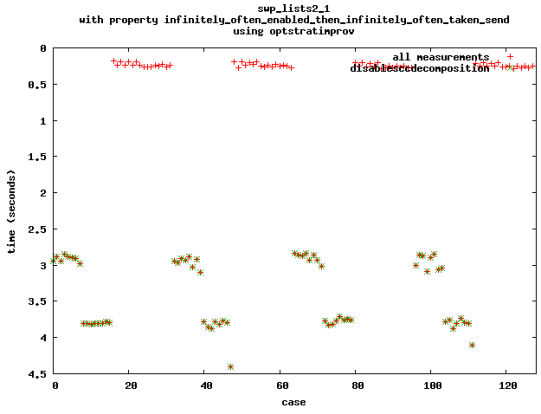


Figure A.39: Optimal strategy improvement

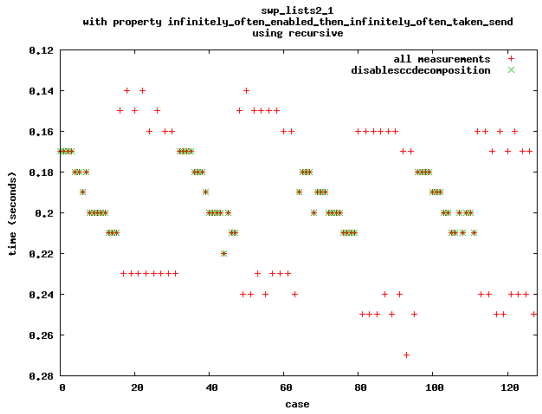


Figure A.40: Recursive

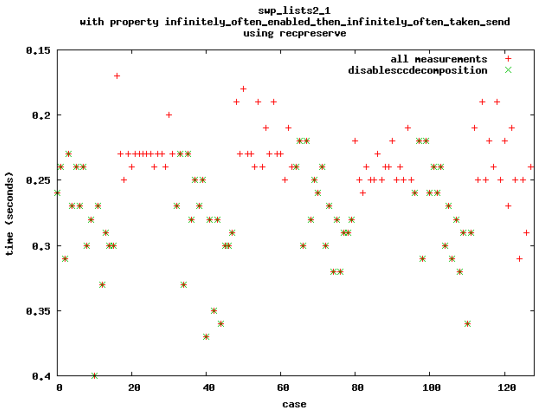


Figure A.41: Recursive preservation

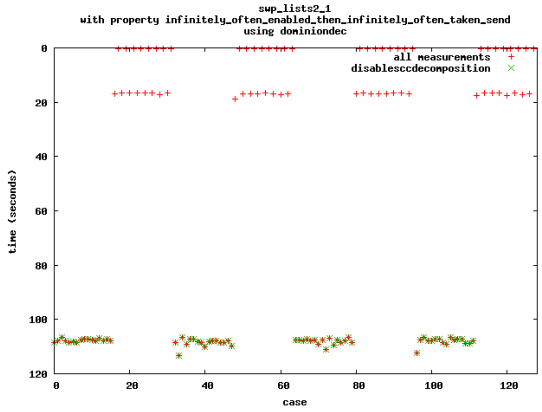


Figure A.42: Dominion decomposition

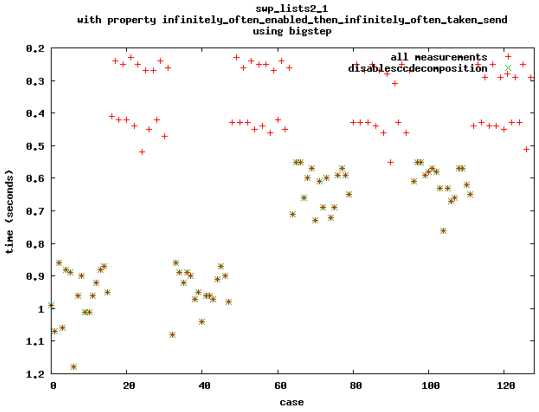


Figure A.43: Bigstep

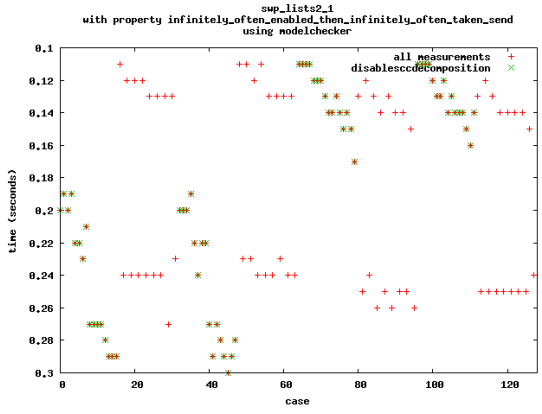


Figure A.44: Model checker

A.2 Solving special games

In order to improve the understanding of the effects of optimisations other than SCC decomposition, we present data where SCC decomposition is always enabled in this section. Furthermore, we show where one of the special game solving (single player or single parity) is enabled.

A.2.1 Deadlock freedom

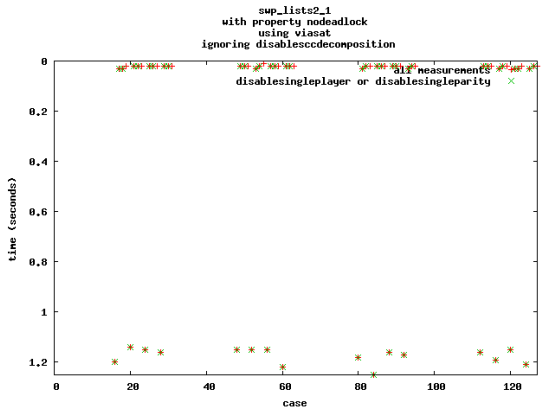


Figure A.45: Viasat

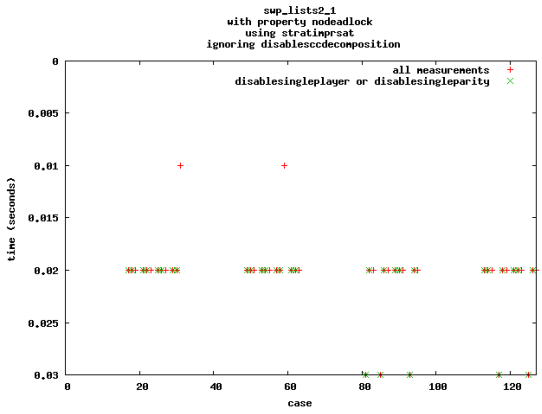


Figure A.46: Stratimprsat

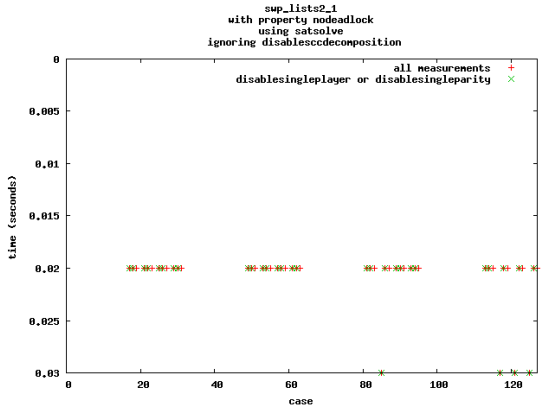


Figure A.47: Satsolve

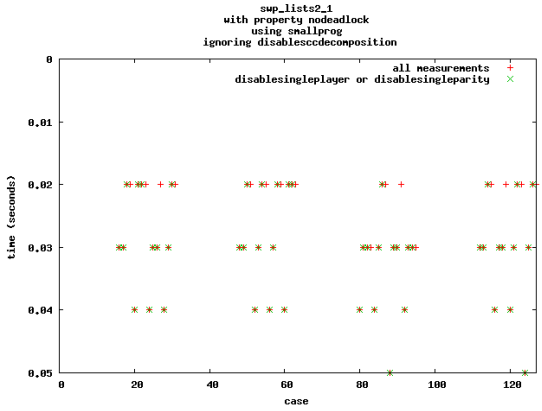


Figure A.48: Small progress measures

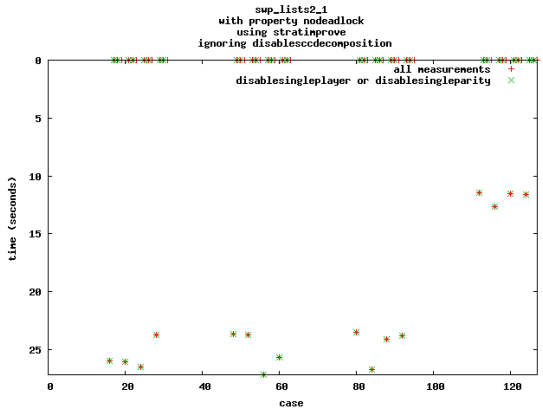


Figure A.49: Strategy improvement

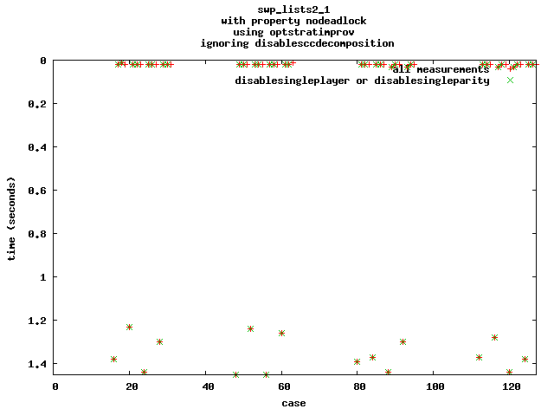


Figure A.50: Optimal strategy improvement

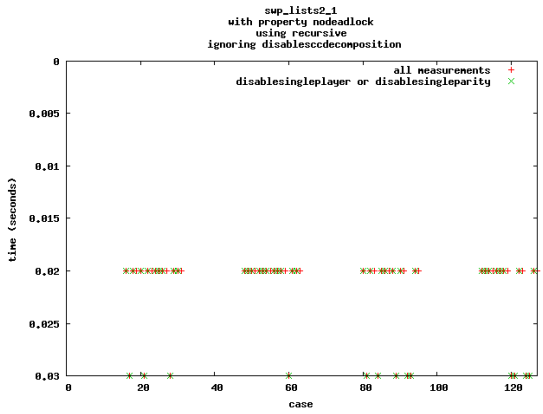


Figure A.51: Recursive

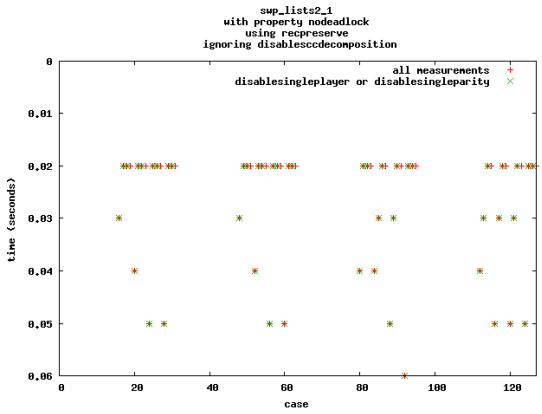


Figure A.52: Recursive preservation

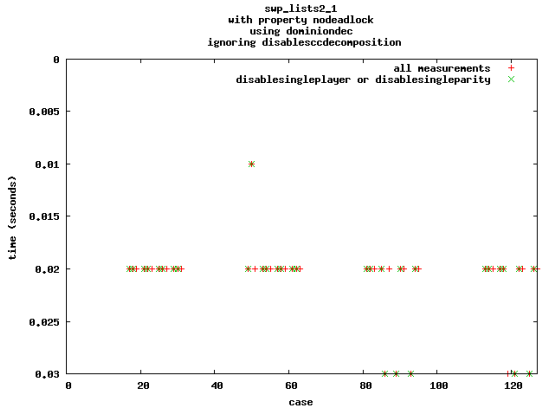


Figure A.53: Dominion decomposition

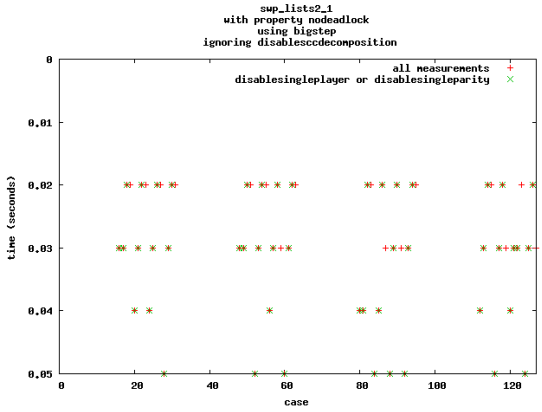


Figure A.54: Bigstep

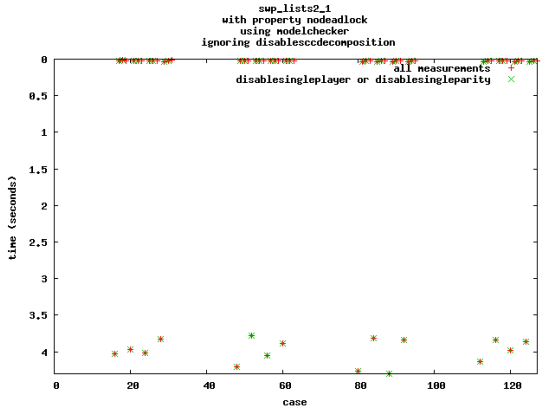


Figure A.55: Model checker

A.2.2 Livelock freedom

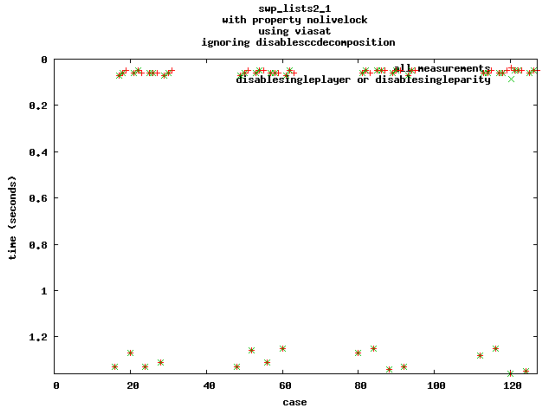


Figure A.56: Viasat

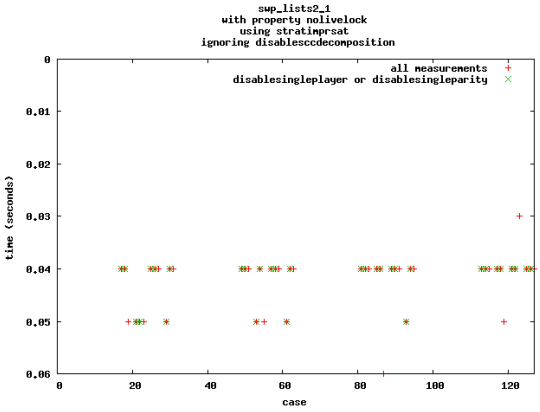


Figure A.57: Stratimprsat

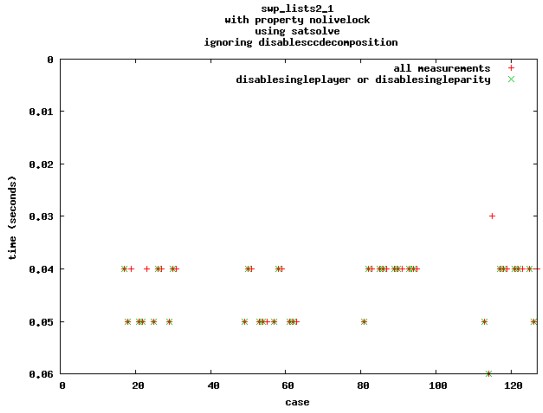


Figure A.58: Satsolve

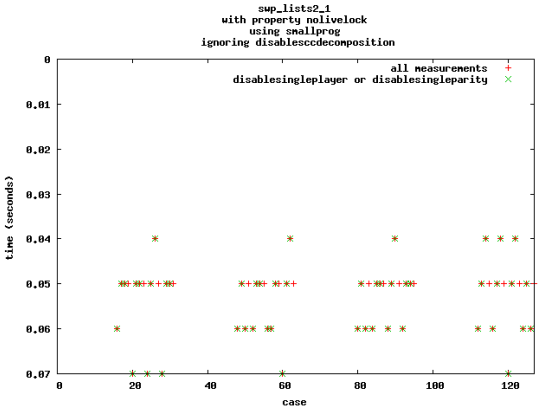


Figure A.59: Small progress measures

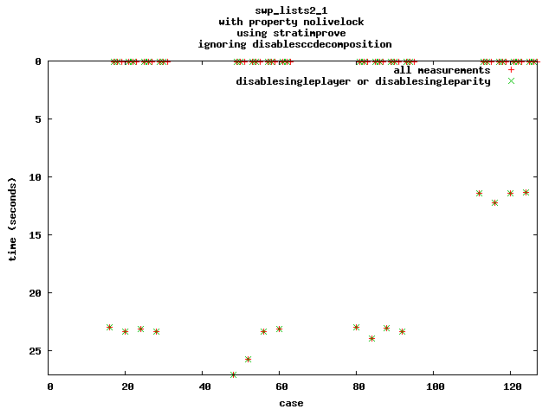


Figure A.60: Strategy improvement

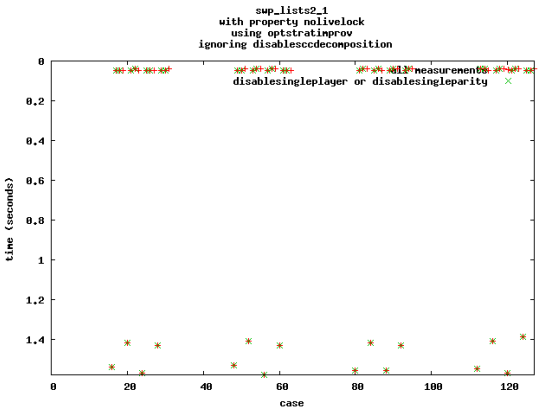


Figure A.61: Optimal strategy improvement

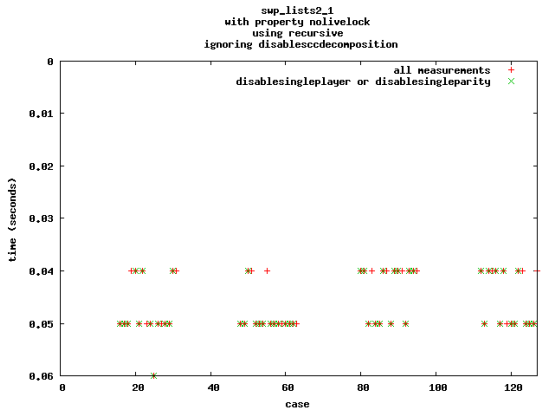


Figure A.62: Recursive

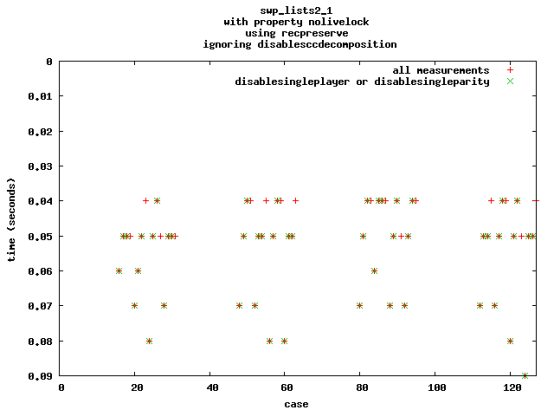


Figure A.63: Recursive preservation

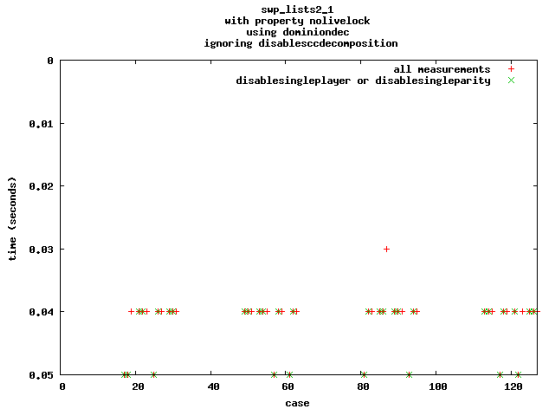


Figure A.64: Dominion decomposition

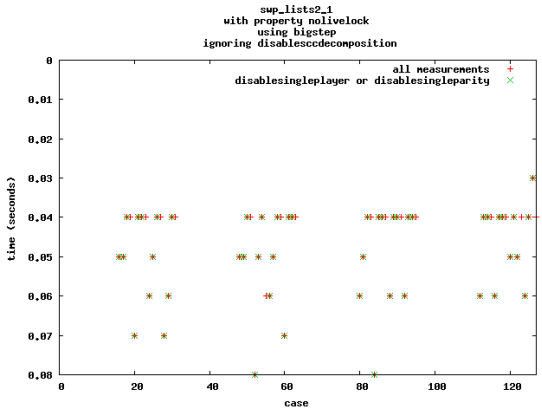


Figure A.65: Bigstep

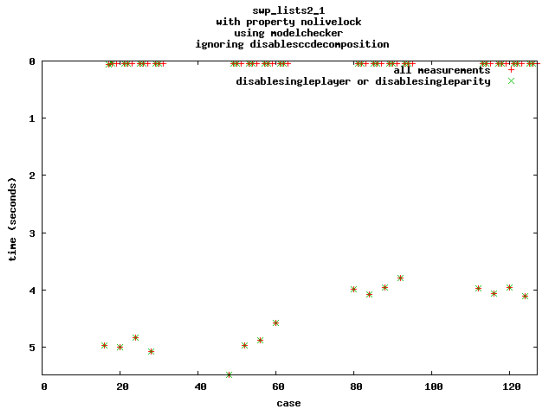


Figure A.66: Model checker

A.2.3 Infinitely often receive

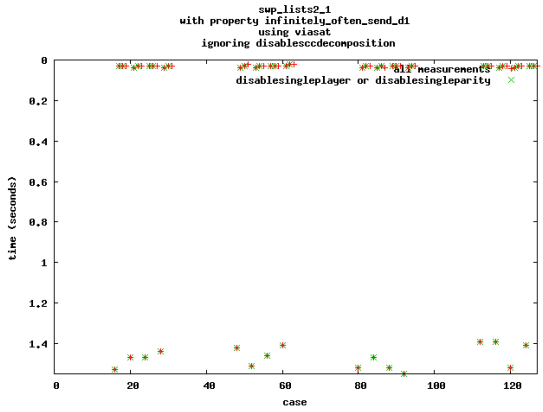


Figure A.67: Viasat

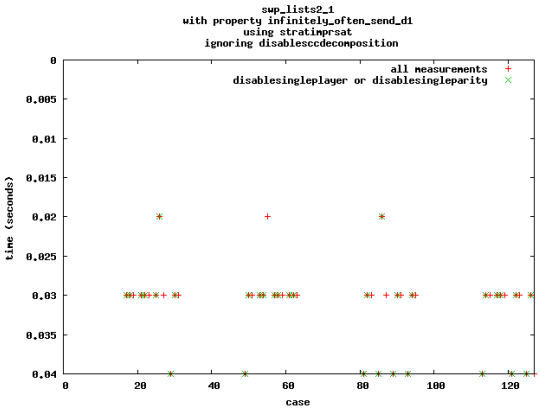


Figure A.68: Stratimprsat

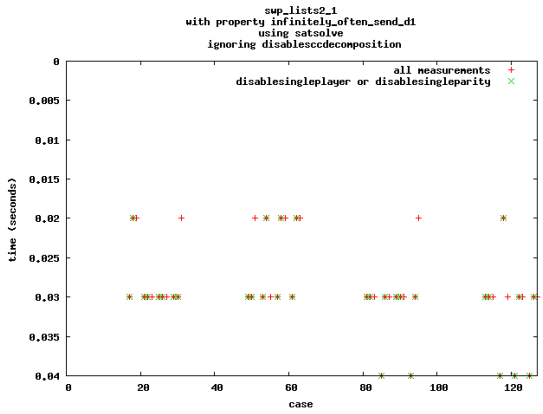


Figure A.69: Satsolve

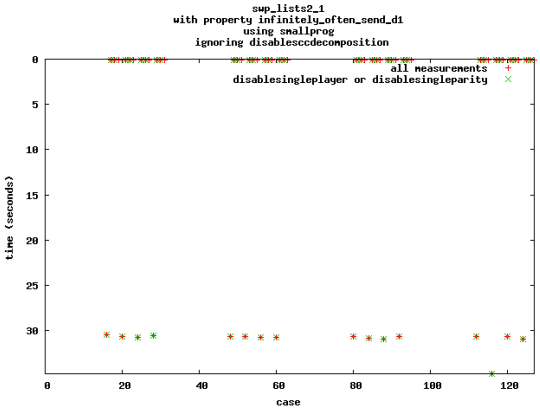


Figure A.70: Small progress measures

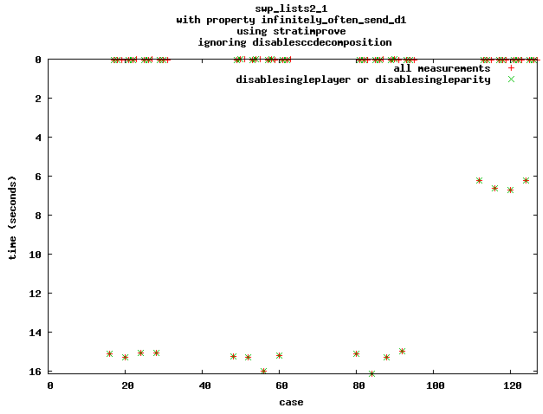


Figure A.71: Strategy improvement

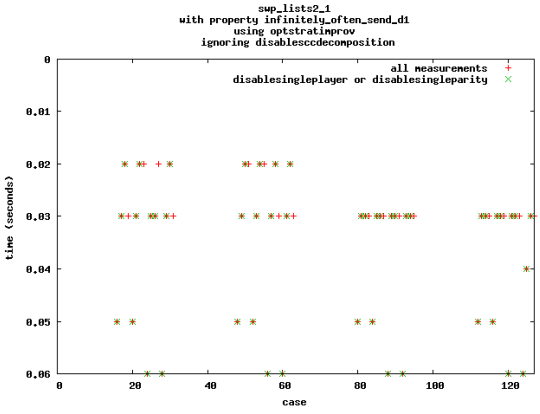


Figure A.72: Optimal strategy improvement

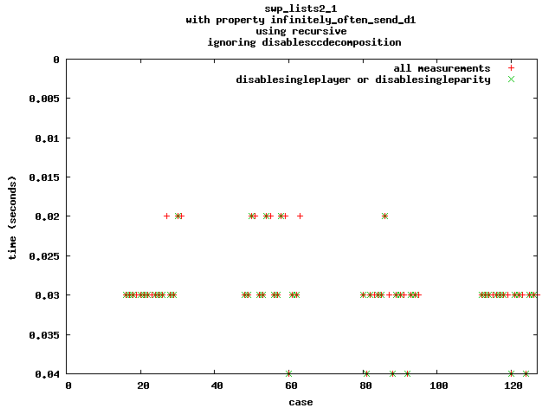


Figure A.73: Recursive

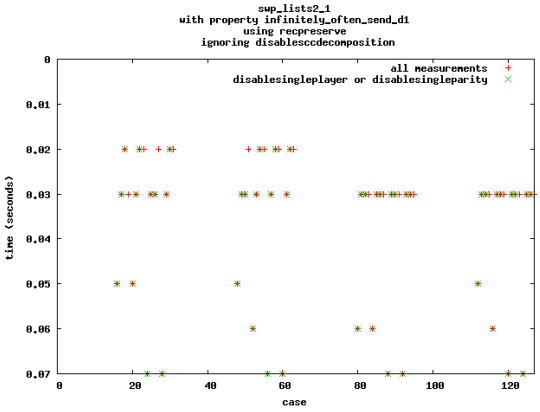


Figure A.74: Recursive preservation

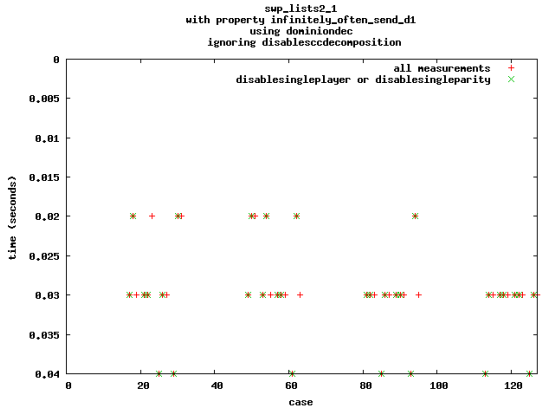


Figure A.75: Dominion decomposition

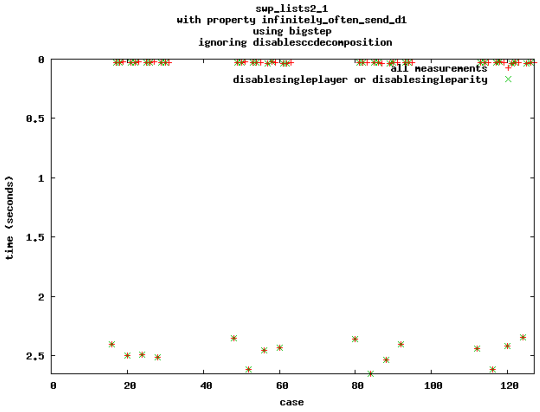


Figure A.76: Bigstep

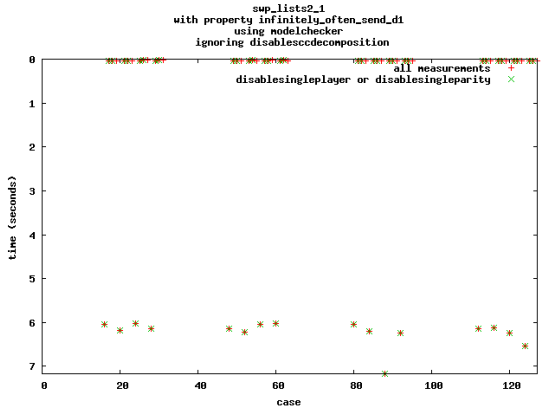


Figure A.77: Model checker

A.2.4 Infinitely often enabled, then infinitely often taken

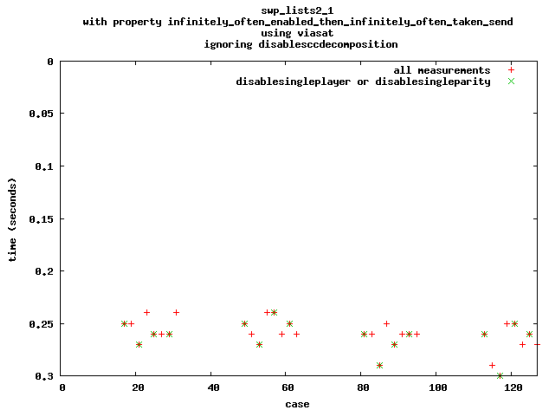


Figure A.78: Viasat

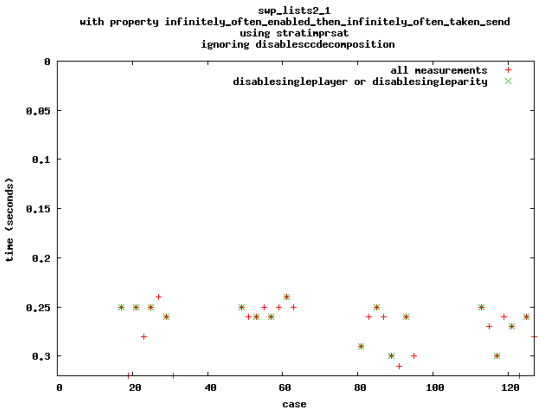


Figure A.79: Stratimprsat

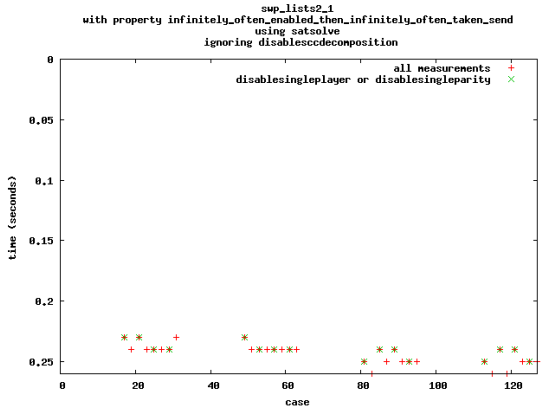


Figure A.80: Satsolve

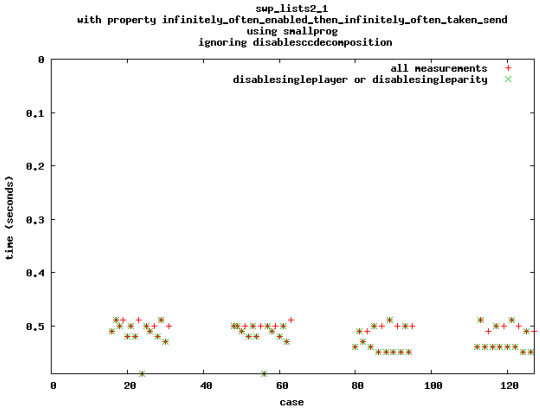


Figure A.81: Small progress measures

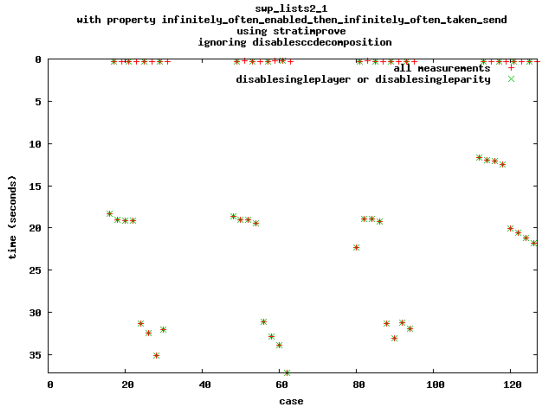


Figure A.82: Strategy improvement

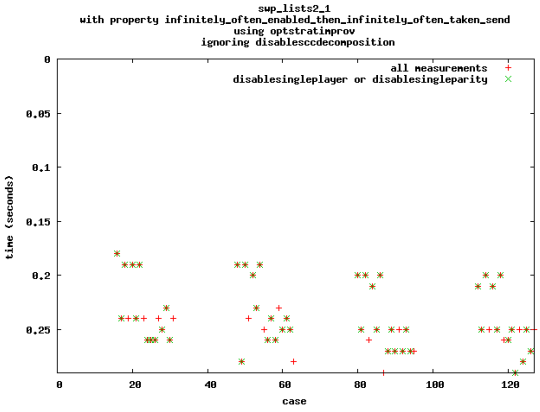


Figure A.83: Optimal strategy improvement

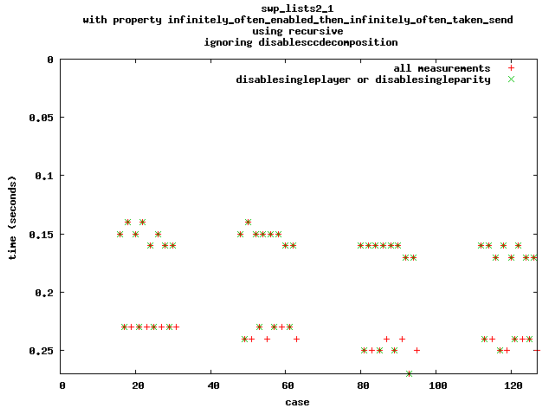


Figure A.84: Recursive

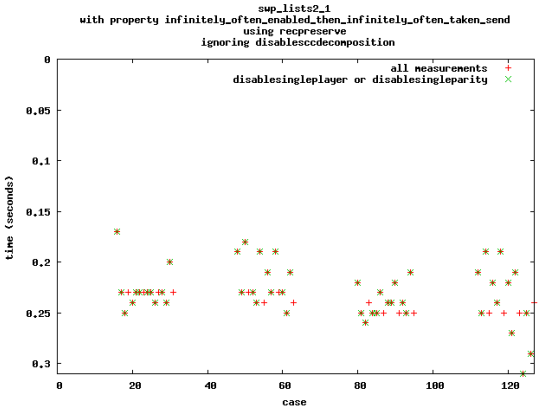


Figure A.85: Recursive preservation

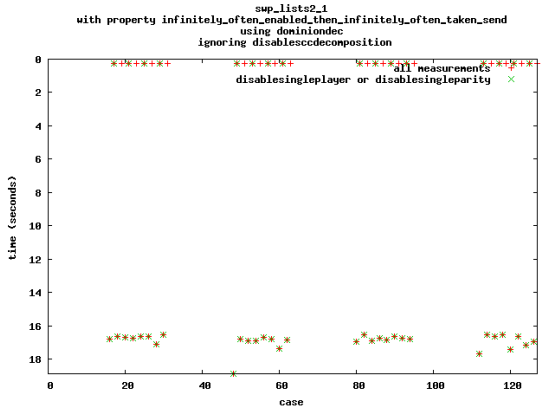


Figure A.86: Dominion decomposition

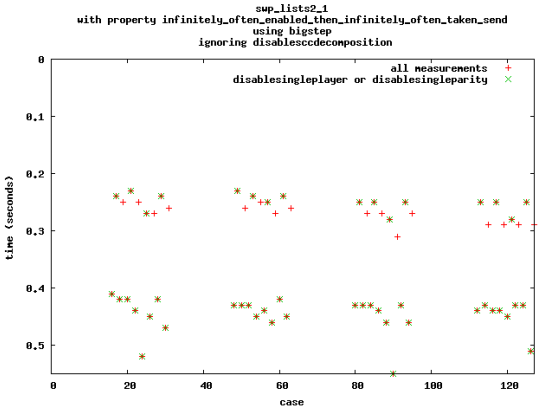


Figure A.87: Bigstep

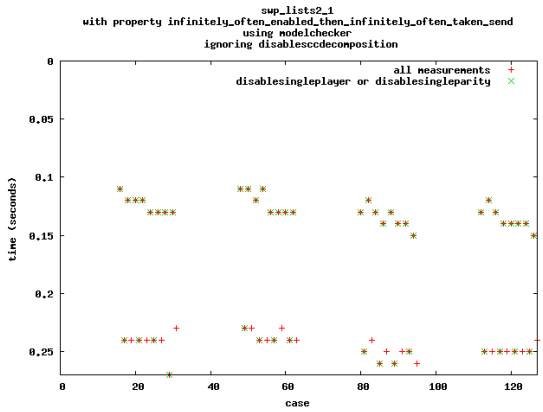


Figure A.88: Model checker

Appendix B

Experimental results for comparison of parity game algorithms

In this appendix we we present the full set of results that were carried out to compare the algorithms as described in Chapter 4 with respect to their performance in practical model checking cases. The analysis was described in Section 5.3.

Because of the large number of experiments that we consider in this section, and the number of different cases that we consider, we use a tabulated presentation of the measurements. Each of the tables considers one specification, and lists for each of the properties, and each of the algorithms, the time required for executing PGSolver with the algorithm. This allows for a relatively quick inventarisation of the best and worst algorithms in each of the cases. Additionally we have created tools to compute the number of times that an algorithm is among the best an worst performing algorithms in order to get a better overall view. In the tables that we present, t/o denotes a timeout of the corresponding run; the time required exceeds 30 minutes. Furthermore ME denotes termination of PGSolver with a memory error.

B.1 Model checking

This section lists the measurements of the algorithms on model checking examples.

Table B.1: abp

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	172	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	254	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	336	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	664	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	1320	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	1976	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
32	2632	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00
nolivelock												
2	394	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	584	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	774	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	1534	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	3054	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	4574	0.01	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
32	6094	0.01	0.01	0.01	0.02	0.02	0.02	0.01	0.01	0.01	0.02	0.02
infinitely often receive d1												
2	178	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	260	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	342	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	670	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	1326	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	1982	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
32	2638	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00
infinitely often receive for all d												
2	353	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	772	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1355	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	5327	0.01	0.01	0.01	0.03	0.02	0.01	0.01	0.02	0.02	0.01	0.02
16	21143	0.12	0.11	0.10	0.19	0.12	0.11	0.11	0.11	0.12	0.11	0.10
24	47455	0.25	0.25	0.25	0.41	0.25	0.33	0.33	0.33	0.35	0.36	0.25
32	84263	0.66	0.50	0.50	1.09	0.50	0.64	0.50	0.50	0.49	0.49	0.50
infinitely often enabled then infinitely often taken receive												
2	1417	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	3160	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
4	5595	0.02	0.02	0.02	0.04	0.02	0.02	0.02	0.02	0.02	0.02	0.02
8	22255	0.17	0.18	0.18	0.31	0.17	0.18	0.16	0.18	0.18	0.17	0.16
16	88791	1.46	1.47	1.46	2.68	1.35	1.47	1.36	1.47	1.49	1.49	1.38
24	199615	5.34	6.14	5.33	11.67	5.75	5.74	6.14	6.16	5.72	5.77	5.73
32	354727	20.84	27.36	17.94	36.44	18.88	17.95	17.92	19.19	18.99	17.93	17.94

Table B.2: abp bw

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	225	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	333	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	441	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	873	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	1737	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	2601	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
32	3465	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
nolivelock												
2	516	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	765	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1014	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	2010	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	4002	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.01	0.01	0.01
24	5994	0.01	0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01
32	7986	0.02	0.02	0.02	0.03	0.03	0.02	0.02	0.02	0.03	0.03	0.02
infinitely often receive d1												
2	233	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	341	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	449	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	881	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	1745	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	2609	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.01
32	3473	0.01	0.01	0.01	0.02	0.01	0.00	0.01	0.01	0.01	0.01	0.01
infinitely often receive for all d												
2	463	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	1015	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	1783	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	7015	0.02	0.02	0.03	0.05	0.02	0.02	0.02	0.02	0.03	0.03	0.03
16	27847	0.15	0.15	0.15	0.26	0.15	0.16	0.15	0.15	0.14	0.16	0.16
24	62503	0.36	0.37	0.37	0.58	0.37	0.37	0.37	0.36	0.38	0.36	0.36
32	110983	0.66	0.65	0.66	1.06	0.66	0.65	0.67	0.65	0.66	0.67	0.67
infinitely often enabled then infinitely often taken receive												
2	1861	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	4150	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
4	7347	0.03	0.03	0.03	0.06	0.03	0.03	0.03	0.03	0.03	0.03	0.03
8	29215	0.22	0.24	0.22	0.43	0.24	0.25	0.25	0.24	0.22	0.24	0.24
16	116535	1.50	1.67	1.64	3.17	1.72	1.53	1.63	1.65	1.50	1.51	1.51
24	261967	5.30	6.12	6.24	10.62	6.12	6.42	5.94	5.96	5.73	6.05	6.00
32	465511	18.24	29.90	18.60	31.74	24.41	16.67	20.01	16.60	27.13	29.07	17.93

Table B.3: cabp

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	2102	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	3390	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
4	4854	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
8	12470	0.03	0.04	0.03	0.04	0.03	0.04	0.04	0.04	0.04	0.04	0.03
16	36150	0.18	0.16	0.16	0.20	0.19	0.16	0.19	0.20	0.16	0.16	0.21
24	71094	0.35	0.29	0.35	0.42	0.29	0.30	0.31	0.36	0.31	0.31	0.36
32	117302	0.75	0.96	0.82	0.97	0.87	0.83	0.91	0.83	0.98	0.99	0.73
nolivelock												
2	4502	0.01	0.01	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.01	0.01
3	7222	0.02	0.02	0.01	0.02	0.01	0.02	0.01	0.02	0.01	0.02	0.02
4	10294	0.02	0.03	0.03	0.03	0.02	0.02	0.03	0.02	0.02	0.03	0.02
8	26102	0.09	0.10	0.11	0.11	0.10	0.10	0.10	0.10	0.11	0.11	0.10
16	74614	0.37	0.38	0.35	0.45	0.39	0.36	0.36	0.38	0.35	0.32	0.37
24	145654	0.69	0.66	0.63	0.80	0.65	0.62	0.62	0.64	0.62	0.71	0.62
32	239222	3.33	2.84	2.89	3.20	2.89	3.24	2.99	3.00	3.14	3.03	2.84
infinitely often receive d1												
2	2200	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	3520	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.00	0.00
4	5016	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.06	0.01	0.01
8	12760	0.04	0.04	0.04	0.09	0.05	0.04	0.05	0.04	0.04	0.04	0.04
16	36696	0.16	0.16	0.16	0.30	0.16	0.16	0.16	0.16	0.16	0.19	0.16
24	71896	0.37	0.29	0.30	0.71	0.37	0.38	0.37	0.37	0.29	0.37	0.29
32	118360	0.60	0.70	0.51	0.93	0.50	0.50	0.50	0.65	0.64	0.65	0.65
infinitely often receive for all d												
2	4397	0.01	0.01	0.01	0.02	0.01	0.01	0.00	0.01	0.01	0.01	0.01
3	10552	0.03	0.03	0.03	0.05	0.03	0.03	0.04	0.03	0.03	0.03	0.03
4	20051	0.06	0.07	0.06	0.12	0.07	0.07	0.07	0.07	0.06	0.07	0.08
8	102047	0.53	0.54	0.41	0.82	0.51	0.42	0.42	0.42	0.42	0.42	0.41
16	587063	3.58	3.02	2.92	5.21	3.47	3.17	3.32	3.50	3.14	3.12	3.33
24	1725391	11.13	8.36	11.09	18.01	8.64	8.36	11.03	8.34	10.90	8.33	8.34
32	3787367	19.27	20.14	20.28	30.84	20.05	19.93	19.33	19.36	19.92	19.91	19.93
infinitely often enabled then infinitely often taken receive												
2	13417	0.07	0.06	0.06	0.13	0.07	0.07	0.07	0.06	0.06	0.06	0.06
3	32338	0.19	0.20	0.21	0.43	0.22	0.20	0.22	0.21	0.21	0.20	0.20
4	61579	0.41	0.45	0.44	0.77	0.45	0.46	0.46	0.45	0.46	0.44	0.46
8	314383	2.71	2.55	2.59	3.89	2.34	2.60	2.56	2.50	2.77	2.68	2.73
16	1811479	20.83	20.61	19.02	35.75	20.62	20.51	20.53	20.47	20.50	20.42	18.79
24	5326879	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
32	11696167	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

Table B.4: swp lists2

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	71094	0.29	0.30	0.30	0.34	0.30	0.30	0.29	0.29	0.30	0.30	0.29
3	269286	1.38	1.38	1.37	1.47	1.38	1.38	1.37	1.38	1.38	1.38	1.39
4	728390	3.82	3.82	3.80	4.34	3.83	3.88	3.83	3.81	3.80	3.80	3.87
5	1614606	10.77	8.72	8.73	9.95	8.76	8.73	8.71	8.79	8.70	8.80	8.73
6	3135606	19.64	19.91	21.03	24.70	19.95	19.79	21.41	20.18	21.41	22.04	20.36
7	5540534	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	9120006	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
nolivelock												
2	113286	0.51	0.50	0.50	0.62	0.50	0.52	0.50	0.52	0.51	0.49	0.51
3	426426	2.14	2.03	2.13	2.66	2.12	2.12	2.13	2.13	2.13	2.13	2.12
4	1149446	6.02	6.01	5.77	6.94	5.76	5.77	5.73	5.75	5.79	5.74	5.79
5	2542506	13.73	13.71	13.67	16.56	13.74	15.42	13.76	13.75	13.71	13.73	13.71
6	4930566	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
7	8703386	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	14315526	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
infinitely often receive d1												
2	78584	0.37	0.36	0.35	0.67	0.37	0.37	0.37	0.37	0.36	0.37	0.36
3	290672	1.58	1.52	1.53	2.91	1.57	1.55	1.54	1.53	1.55	1.55	1.54
4	774472	4.21	4.21	4.19	7.73	4.21	4.33	4.20	4.20	4.19	4.21	4.19
5	1699208	9.68	9.74	10.97	19.79	9.74	9.65	11.15	10.95	9.72	9.70	9.70
6	3275576	24.01	24.07	22.13	41.77	23.18	23.14	22.28	22.59	22.22	22.64	22.28
7	5755744	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
8	9433352	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
infinitely often receive for all d												
2	157165	0.80	0.79	0.80	1.31	0.80	0.81	0.81	0.81	0.81	0.81	0.81
3	872008	4.83	4.83	4.83	9.58	4.98	4.88	4.83	4.72	5.01	4.90	4.83
4	3097875	19.04	19.13	19.01	31.21	18.87	19.13	18.97	19.09	18.98	19.96	19.01
5	8496022	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
infinitely often enabled then infinitely often taken receive												
2	457833	13.10	13.42	13.55	27.90	12.75	11.76	12.05	11.69	12.17	11.75	11.83
3	2574406	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
4	9223947	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
5	25444512	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

Table B.5: swp lists3

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	1187910	6.52	6.56	6.65	8.09	6.53	6.52	6.44	6.44	6.67	6.43	6.47
3	8922156	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
nolivelock												
2	1858086	10.29	10.27	10.27	11.85	10.45	10.36	10.41	10.25	10.44	10.57	10.47
3	13880328	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
infinitely often receive d1												
2	1329992	8.52	8.24	8.74	14.35	8.64	8.44	8.47	8.37	8.82	8.51	8.31
3	9712070	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
infinitely often receive for all d												
2	2659981	16.53	17.76	17.10	30.59	16.81	17.99	17.12	17.16	16.95	17.28	17.74

Table B.6: brp

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
						nodeadlock						
2	9040	0.04	0.05	0.04	0.06	0.04	0.04	0.04	0.04	0.04	0.04	0.04
3	22722	0.14	0.14	0.14	0.18	0.14	0.17	0.14	0.14	0.14	0.14	0.14
4	46448	0.34	0.34	0.34	0.41	0.34	0.42	0.35	0.43	0.35	0.35	0.42
8	298192	2.89	2.87	2.87	4.23	2.87	3.03	2.86	2.85	2.86	2.86	2.89
16	2141072	28.68	28.47	32.18	ME	28.40	25.43	26.20	25.69	25.46	31.99	26.02
						nolivelock						
2	22272	0.09	0.09	0.10	0.12	0.10	0.09	0.09	0.10	0.10	0.09	0.09
3	55986	0.29	0.29	0.28	0.36	0.29	0.29	0.29	0.29	0.29	0.28	0.29
4	114448	0.66	0.65	0.65	0.80	0.64	0.69	0.68	0.70	0.65	0.67	0.67
8	734736	5.02	5.03	6.10	6.10	5.27	5.24	5.10	5.25	5.72	5.81	5.82
16	5275408	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

Table B.7: onebit

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
						nodeadlock						
2	550086	2.68	2.66	2.67	2.99	2.68	2.68	2.68	2.67	2.68	2.68	2.67
3	1978998	10.52	10.47	10.52	11.69	10.46	10.63	10.49	10.61	10.70	12.12	10.59
4	5140230	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
						nolivelock						
2	1044870	5.20	5.20	5.23	5.97	5.20	5.21	5.19	5.23	5.23	5.22	5.23
3	3724710	21.76	21.73	21.61	25.27	21.88	21.49	21.41	21.42	21.86	21.76	21.66
4	9616902	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

Table B.8: 1394-fin

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
517118	4.41	4.48	4.50	5.33	4.50	4.46	4.58	4.49	4.47	4.48	4.45
					nolivelock						
1120841	6.91	7.40	6.90	10.67	7.37	6.92	6.94	6.96	6.37	6.87	7.02

Table B.9: chatbox

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
589830	2.64	2.70	2.61	3.23	2.73	2.60	2.69	2.66	2.85	2.75	2.65
					nolivelock						
1245190	5.79	5.90	5.62	6.33	6.24	5.89	5.64	6.06	6.36	6.05	5.97

Table B.10: dining 10

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
1140887	4.23	4.43	4.34	7.34	4.31	4.38	4.15	4.34	4.31	4.56	4.39
					nolivelock						
1604236	9.82	9.65	9.40	11.04	9.45	10.04	9.59	9.57	10.10	10.10	9.42

Table B.11: dining3 cs

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
146	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
254	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
663	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00
					nostuffing						
638	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.01

Table B.12: dining3 cs seq

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
111	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
219	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
529	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostuffing						
533	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.13: dining3

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
172	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
809	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
1594	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostuffing						
2134	0.09	0.09	0.09	0.19	0.08	0.08	0.09	0.09	0.08	0.08	0.09

Table B.14: dining3 ns

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
63	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
243	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
550	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostuffing						
613	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table B.15: dining3 ns seq

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
108	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
212	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
512	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostuffing						
520	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.16: dining3 schedule

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
132	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
267	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostarvation						
639	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nostuffing						
652	0.03	0.03	0.03	0.07	0.03	0.03	0.03	0.03	0.03	0.03	0.03

Table B.17: dining3 schedule seq

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock											
132	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nolivelock											
267	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nostarvation											
639	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nostuffing											
652	0.03	0.03	0.03	0.07	0.03	0.04	0.03	0.03	0.07	0.03	0.04

Table B.18: dining3 seq

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock											
326	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nolivelock											
603	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nostarvation											
1554	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
nostuffing											
1516	0.07	0.07	0.07	0.14	0.07	0.07	0.07	0.07	0.07	0.07	0.07

Table B.19: domineering

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock											
2518019	12.64	13.24	12.57	15.27	14.98	13.50	12.87	13.07	12.36	13.12	12.71
nolivelock											
3883970	21.54	19.47	20.96	25.00	20.06	19.84	20.92	19.39	20.11	19.17	19.44
eventually player1 or player2 wins											
2518019	10.92	11.05	11.10	14.68	11.14	11.08	11.05	11.07	11.90	11.84	11.17
player1 can win											
2518019	12.84	12.42	12.21	15.84	12.14	12.20	12.23	12.19	12.81	15.15	12.17
player2 can win											
2518019	15.09	12.37	13.65	17.84	12.11	13.16	12.41	12.12	12.40	12.45	12.14

Table B.20: goback

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
502505	1.23	1.25	1.21	1.25	1.23	1.23	1.27	1.22	1.22	1.23	1.22
					nolivelock						
505505	1.28	1.24	1.26	1.29	1.24	1.24	1.25	1.24	1.24	1.24	1.26

Table B.21: leader5

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
1527	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
3439	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.22: leader8

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
19139	0.04	0.04	0.04	0.07	0.04	0.04	0.04	0.04	0.04	0.04	0.05
					nolivelock						
41842	0.11	0.11	0.10	0.13	0.11	0.10	0.10	0.11	0.10	0.11	0.10

Table B.23: lift3-final

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
14236	0.05	0.05	0.04	0.06	0.05	0.05	0.05	0.05	0.05	0.05	0.04
					nolivelock						
28032	0.10	0.09	0.10	0.14	0.10	0.10	0.10	0.10	0.09	0.09	0.10

Table B.24: lift3-init

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
150213	0.60	0.60	0.60	0.82	0.60	0.60	0.63	0.61	0.60	0.60	0.60
					nolivelock						
259154	1.14	1.14	1.14	1.45	1.15	1.14	1.14	1.14	1.13	1.13	1.14

Table B.25: mpsu

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
	nodeadlock										
171	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	nolivelock										
327	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.26: producer consumer

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
	nodeadlock										
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	nolivelock										
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.27: snake

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
	nodeadlock										
171343	1.10	1.09	1.08	1.36	1.08	1.08	1.07	1.11	1.08	1.08	1.07
	nolivelock										
399349	2.11	2.09	2.09	2.66	2.11	2.11	2.20	2.10	2.11	2.11	2.11
	black can win										
171343	1.00	0.99	0.99	1.61	0.99	0.99	0.99	0.99	0.97	1.00	1.00
	eventually white or black wins										
171343	0.93	0.93	0.94	1.23	0.92	0.94	0.93	0.94	0.94	0.93	0.94
	white can win										
171343	1.00	1.00	0.99	1.65	1.00	1.00	1.00	0.99	0.99	1.00	1.00

Table B.28: tree

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
	nodeadlock										
2568	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	nolivelock										
5130	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table B.29: scheduler

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
nodeadlock												
2	38	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	116	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	344	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	968	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	2600	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	16904	0.05	0.05	0.05	0.06	0.04	0.04	0.04	0.04	0.04	0.05	0.05
nolivelock												
2	77	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	227	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	635	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	1691	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	4331	0.01	0.00	0.01	0.01	0.01	0.01	0.00	0.01	0.00	0.00	0.00
8	26123	0.08	0.08	0.08	0.10	0.08	0.09	0.08	0.08	0.09	0.08	0.09

Table B.30: trains1

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
42	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.31: trains2

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
200	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.32: trains3

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
202	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.33: trains4

	Solving times										
size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
					nodeadlock						
165	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
					nolivelock						
356	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

B.2 Equivalence checking

This section lists the measurements of the algorithms on equivalence checking examples.

Table B.34: Branching bisimilarity abp – abp bw

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	4509	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	6780	0.02	0.01	0.01	0.03	0.01	0.01	0.01	0.01	0.01	0.01	0.01
4	9065	0.02	0.02	0.02	0.03	0.02	0.02	0.02	0.02	0.02	0.02	0.02
5	23664	0.12	0.09	0.09	0.14	0.10	0.09	0.12	0.12	0.10	0.11	0.09
6	28539	0.11	0.12	0.12	0.17	0.12	0.11	0.12	0.12	0.13	0.11	0.12
7	33462	0.15	0.16	0.15	0.20	0.14	0.14	0.15	0.15	0.15	0.14	0.16
8	18345	0.06	0.06	0.06	0.09	0.06	0.06	0.06	0.06	0.06	0.06	0.06
9	20700	0.07	0.08	0.08	0.15	0.08	0.07	0.07	0.07	0.07	0.08	0.07
10	23069	0.08	0.08	0.08	0.12	0.08	0.08	0.08	0.08	0.08	0.08	0.08
16	37577	0.16	0.16	0.16	0.25	0.16	0.16	0.16	0.16	0.16	0.15	0.16
20	101829	0.52	0.52	0.55	0.70	0.53	0.55	0.57	0.55	0.54	0.54	0.54
24	124497	0.68	0.63	0.66	0.83	0.63	0.64	0.63	0.64	0.63	0.63	0.64
28	147933	0.80	0.80	0.78	1.06	0.79	0.79	0.85	0.80	0.80	0.81	0.79
32	172137	1.11	0.92	1.11	1.35	1.01	0.91	0.91	1.01	1.01	1.01	0.91

Table B.35: Branching bisimilarity abp – cabp

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	205846	t/o	ME	ME	1.97	t/o	t/o	1.53	2.15	ME	1.81	ME
3	334038	t/o	ME	ME	3.61	t/o	t/o	2.59	3.89	ME	3.06	ME
4	479158	t/o	ME	ME	5.27	t/o	t/o	3.83	5.36	ME	4.37	ME
5	641206	ME	ME	ME	8.88	t/o	t/o	6.34	7.27	t/o	5.93	ME
6	820182	ME	ME	ME	10.30	t/o	t/o	6.77	11.70	ME	7.77	ME
7	1016086	ME	ME	ME	11.08	t/o	t/o	8.78	16.03	ME	11.03	ME
8	1228918	ME	ME	ME	13.92	ME	t/o	10.26	14.32	ME	11.91	ME
9	1458678	ME	ME	ME	16.08	ME	t/o	12.58	17.85	t/o	14.81	ME
10	1705366	ME	ME	ME	19.00	ME	t/o	15.26	21.19	ME	17.61	ME
16	3540982	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
20	5103286	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
24	6936438	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
28	9040438	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME
32	11415286	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME	ME

Table B.36: Branching bisimilarity abp – oneplacebuffer

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	844	0.12	1573.34	ME	0.00	0.62	0.03	0.00	0.00	t/o	0.00	0.01
3	1434	0.46	t/o	ME	0.00	3.45	0.08	0.00	0.00	t/o	0.01	0.03
4	2140	0.80	t/o	ME	0.01	12.18	0.16	0.00	0.01	t/o	0.01	0.07
5	2962	3.37	ME	ME	0.02	32.52	0.32	0.01	0.01	t/o	0.02	0.13
6	3900	3.08	ME	ME	0.03	80.73	0.54	0.01	0.02	t/o	0.03	0.24
7	4954	5.59	ME	ME	0.04	158.65	0.81	0.01	0.02	t/o	0.04	0.34
8	6124	11.98	ME	ME	0.05	306.73	1.19	0.02	0.04	t/o	0.05	0.51
9	7410	85.56	ME	ME	0.07	540.73	1.74	0.03	0.05	t/o	0.06	0.74
10	8812	38.61	ME	ME	0.09	898.05	2.35	0.03	0.06	t/o	0.08	1.06
16	19660	462.46	ME	ME	0.26	t/o	11.40	0.11	0.16	t/o	0.23	5.05
20	29212	t/o	ME	ME	0.47	t/o	26.71	0.19	0.27	t/o	0.38	11.92
24	40620	t/o	ME	ME	0.65	t/o	42.99	0.28	0.39	t/o	0.55	26.33
28	53884	t/o	ME	ME	0.81	t/o	71.54	0.38	0.54	t/o	0.90	48.62
32	69004	t/o	ME	ME	1.12	t/o	106.00	0.50	0.71	t/o	1.08	113.75

Table B.37: Branching bisimilarity abp bw – cabp

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	36492	0.13	0.14	0.14	0.20	0.14	0.13	0.13	0.14	0.13	0.13	0.14
3	55149	0.22	0.22	0.21	0.38	0.22	0.22	0.21	0.21	0.22	0.22	0.22
4	74094	0.30	0.30	0.29	0.44	0.29	0.30	0.29	0.29	0.30	0.30	0.29
5	218352	1.24	8.04	8.19	1.38	1.17	1.29	1.29	1.29	1.29	1.29	1.15
6	264414	1.48	9.61	9.17	1.93	1.55	1.51	1.37	1.37	1.53	1.52	1.52
7	311276	1.84	11.07	10.75	2.03	1.74	1.71	1.72	1.69	1.72	1.71	1.71
8	152754	0.68	0.67	0.67	1.02	0.68	0.67	0.68	0.68	0.68	0.68	0.67
9	173139	0.77	0.76	0.76	1.17	0.75	0.76	0.76	0.76	0.75	0.76	0.76
10	193812	0.91	0.90	0.90	1.38	0.95	0.91	0.95	0.91	0.91	0.90	0.90
16	323898	1.51	1.50	1.50	2.61	1.50	1.51	1.49	1.50	1.50	1.51	1.53
20	993282	6.06	31.56	32.26	7.00	5.62	5.70	5.53	5.54	5.55	5.52	5.51
24	1230330	8.70	39.57	40.12	9.01	8.23	7.57	7.85	7.53	7.44	7.34	7.45
28	1480178	11.05	47.22	49.41	11.71	9.74	8.82	9.63	9.72	8.80	9.68	9.61
32	1742826	11.46	53.56	55.56	13.19	10.81	10.71	10.65	10.93	10.71	10.51	11.04

Table B.38: Branching bisimilarity abp bw – oneplacebuffer

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	237	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	375	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	529	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	1489	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	1935	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	2431	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	1305	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	1539	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	1789	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	3625	0.00	0.01	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.01	0.00
20	13429	0.04	0.04	0.04	0.07	0.04	0.04	0.04	0.04	0.04	0.04	0.04
24	18513	0.07	0.06	0.07	0.12	0.07	0.07	0.06	0.07	0.07	0.06	0.07
28	24397	0.09	0.09	0.08	0.14	0.09	0.08	0.09	0.10	0.08	0.09	0.09
32	31081	0.13	0.12	0.11	0.19	0.13	0.12	0.11	0.12	0.11	0.12	0.11

Table B.39: Branching bisimilarity cabp – oneplacebuffer

		Solving times										
$\|M\|$	size	vs	is	ss	sp	si	os	re	rp	dd	bs	mc
2	12022	0.04	0.04	0.04	0.05	0.04	0.03	0.04	0.03	0.04	0.04	0.04
3	21470	0.08	0.08	0.08	0.10	0.07	0.07	0.07	0.07	0.08	0.08	0.08
4	33270	0.14	0.14	0.13	0.15	0.13	0.13	0.14	0.13	0.13	0.13	0.13
5	47422	0.19	0.22	0.21	0.26	0.20	0.22	0.22	0.21	0.20	0.21	0.22
6	63926	0.30	0.31	0.32	0.33	0.31	0.30	0.28	0.31	0.28	0.31	0.28
7	82782	0.37	0.37	0.37	0.43	0.37	0.36	0.37	0.37	0.39	0.37	0.37
8	103990	0.48	0.49	0.49	0.57	0.48	0.48	0.49	0.49	0.48	0.49	0.48
9	127550	0.60	0.61	0.66	0.71	0.61	0.61	0.61	0.61	0.60	0.61	0.61
10	153462	0.73	0.73	0.74	0.86	0.73	0.74	0.74	0.73	0.73	0.74	0.73
16	358326	1.78	1.77	1.86	2.31	1.77	1.77	1.78	1.80	1.78	1.78	1.77
20	541942	2.81	2.81	2.80	3.22	2.79	2.88	2.78	2.84	2.81	2.78	2.84
24	763190	4.26	4.27	4.12	4.84	4.12	4.19	4.19	4.05	4.11	4.33	4.00
28	1022070	5.56	6.09	6.15	7.07	6.11	5.47	6.10	6.06	6.11	6.16	5.53
32	1318582	7.89	7.82	7.66	8.69	7.72	7.51	7.81	7.55	7.54	7.60	7.58

Appendix C

Modal formulae in mCRL2 syntax

This chapter lists the modal formulae that have been used in the experiments in mCRL2 syntax.

Absence of deadlock (5.1):

```
[true*]<true>true
```

Absence of livelock (5.2):

```
[true*]mu X.[tau]X
```

Message $d1$ can be received through $r1$ infinitely often (5.3):

```
nu X. mu Y. (<r1(d1)>X || <!r1(d1)>Y)
```

All messages d can be received through $r1$ infinitely often (5.4):

```
forall d:D . nu X. mu Y. (<r1(d)>X || <!r1(d)>Y)
```

For all messages d it holds that if a receive of d is infinitely often enabled, then it is infinitely often taken (5.5):

```
forall d:D . ([true*] nu X. mu Y. nu Z.
               ([r1(d)]X && ([r1(d)]false || [!r1(d)]Y) && [!r1(d)]Z))
```

No starvation (5.6):

```
[true*](forall p: Phil. mu Y. ([!eat(p)]Y && <true>true))
```

No stuffing (5.7):

```
forall p:Phil. nu X. mu Y. [eat(p)]Y && [!eat(p)]X
```

Eventually player 1 or player 2 wins (5.8):

```
mu X.<Player1Wins || Player2Wins>true || [true]X
```

Player 1 can win (5.9):

```
mu X. <Player1Wins>true || <true>X
```