Partial-Order Reduction for Parity Games with an Application on Parameterised Boolean Equation Systems (Technical Report)

Thomas Neele, Tim A.C. Willemse, and Wieger Wesselink

Eindhoven University of Technology, The Netherlands

Abstract. Partial-order reduction (POR) is a well-established technique to combat the problem of state-space explosion. Most approaches in literature focus on Kripke structures or labelled transition systems and preserve a form of stutter/weak trace equivalence or weak bisimulation. Therefore, they are at best applicable when checking weak modal mu-calculus. We propose to apply POR on parity games, which can encode the combination of a transition system and a temporal property. Our technique allows one to apply POR in the setting of mu-calculus model checking. We show with an example that the reduction achieved on parity games can be significantly larger. Furthermore, we identify and repair an issue where stubborn sets do not preserve stutter equivalence.

1 Introduction

In the field of formal methods, model checking [2] is a popular technique to analyse the behaviour of concurrent processes. However, the arbitrary interleaving of these parallel processes can cause an exponential blowup, which is known as the *state-space explosion* problem. Several approaches have been identified to alleviate this issue, by reducing the state space *on-the-fly*, *i.e.*, while generating it. Two established techniques are *symmetry reduction* [10] and *partial-order reduction* (POR) [7,18,20]. Whereas symmetry reduction can only be applied to systems that contain several copies of a component, POR also applies to heterogeneous systems. However, a major drawback of POR is that most variants at best preserve only a fragment of a given logic, such as LTL or CTL* without the next operator (LTL_{-X}/CTL^*_{-X}) [6] or the weak modal mu-calculus [19]. Furthermore, the variants of POR that preserve a branching logic impose significant restrictions on the reduction by only allowing the prioritisation of exactly one action at a time. This decreases the amount of reduction achieved.

In this paper, we address these shortcomings by applying POR on parity games. A parity game is a directed graph with decorations on the nodes that is played between two players, *even* ($\Diamond$) and *odd* ($\Box$). An application of parity games is encoding a model checking question: a combination of a model, in the form of a labelled transition system (LTS) or Kripke structure (KS), and a formal property, formulated in the modal mu-calculus [12]. In such games, every node v represents the combination of a state s from the transition system and

a (sub)formula φ. Under a typical encoding, player $\Diamond$ wins in v if and only if φ holds in s. In the context of model checking, parity games suffer from the same state-space explosion that models do. Exploring the state space of a parity game under POR can be a very effective way to address this. Our contributions are as follows:

- We propose conditions under which a parity game can be reduced. We prove that the reduction is correct, *i.e.*, it preserves the winning player of the parity game (Theorem 1).
- We show with an example that one of the POR conditions, called **D1** in [23], often found in literature contains a subtle mistake. When applied to LTSs or KSs, this means that LTL_{-X} is not necessarily preserved. This omission is addressed by the strengthened version of condition **D1** that we propose.
- We identify improvements for the reduction by investigating the typical structure of a parity game that encodes a model checking question.
- We give details of a possible implementation, using *parameterised Boolean equation systems* (PBESs) [9] as a high-level representation. The implementation is based on the generic framework of [14], but extends it with support for non-determinism.

The proposed approach has two benefits over traditional POR techniques that operate on LTSs or KSs. First, it enables the use of partial-order reduction in the setting of mu-calculus model checking. This improves on previous works, since they only support the weak mu-calculus [19] or LTL_{-X} [23]/CTL^*_{-X} [6], which are even less expressive logics. Furthermore, the conditions that we propose are strictly weaker than those necessary to preserve the branching structure of an LTS or KS [6,19].

The rest of the paper is structured as follows. Section 2 compares the current work to existing literature. Then, we introduce several basic concepts in Section 3. Our main ideas are presented in Section 4 and further improved in Section 5. Section 6 gives an introduction to PBESs and explains how they relate to parity games. Details of a possible implementation with PBESs are given in Section 7 and the results of a first experiment are presented in Section 8. Finally, Section 9 concludes and provides possible directions for future work.

2 Related Work

There are several related works on partial-order reduction for branching-time logics. First, Groote and Sellink [8] propose several forms of confluence reduction and prove which behavioural equivalences are preserved. In confluence reduction, one tries to identify inert τ-transitions that do not influence the rest of the system. The state space can be reduced by prioritising those transitions.

The work of Ramakrishna and Smolka [19] is also based on inert τs and prioritisation. What they propose coincides with strong confluence from [8], and thus their algorithm preserves weak bisimulation and the corresponding logic weak modal mu-calculus. Since they only present an algorithm that is tightly

integrated into a local model checking procedure, it is not immediately clear what the essence of their idea is. They do not present the concept of a reduction function or conditions thereon separately.

Similar ideas are presented by Gerth *et al.* in [6], but their setting of Kripke structures matches ours more closely. Their approach is based on the *ample set* method [18] and preserves a relation they call visible bisimulation, which is more commonly known as divergence-preserving branching bisimulation. As a result, they preserve CTL_{-X}.

POR has also been applied to other models of computation, such as probabilistic processes [1] and real-time models [3].

To obtain more insight into the difference between POR applied on processes or on parity games, we consider the ideas of Gerth *et al.* [6] in more detail. Their conditions **C1**-**C3** preserve LTL_{-X} and are approximately equal to our conditions. However, to preserve the branching structure, they introduce the following *singleton proviso*:

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

This extra condition can severely impact the amount of reduction achieved, as shown by the following example.

Example 1. Consider the process P, defined as follows.

$$A(n{:}\mathbb{N}) = (n \neq 0) \to (a + a') \cdot A(n-1)$$
$$B(n{:}\mathbb{N}) = (n \neq 0) \to (b + b') \cdot B(n-1)$$
$$P = \nabla_{\{a,a',b,b'\}}(A(N) \parallel B(N))$$

Where N is a large natural number. Due to **C4**, neither $\{a, a'\}$ nor $\{b, b'\}$ is a valid stubborn set. However, when constructing a PBES for the formula $[\mathit{true}^*]\langle \mathit{true}^*.a\rangle \mathit{true}$, we obtain a parity game with a very similar shape that *can* be reduced by prioritising transitions that correspond to b or b'. Note that this mu-calculus formula cannot be represented in LTL. Therefore, condition **C4** is in general required to preserve its validity. □

Several optimisations for CTL model checking under POR are proposed in [15]. They show that condition **C4** is not required to preserve certain common classes of CTL formulas. This can significantly improve the reduction, as the above example also demonstrates.

3 Preliminaries

In this section, we give the standard definition of a *parity game* and its related concepts.

Definition 1. *A* parity game *is a directed graph* $G = (V, \to, \Omega, \mathcal{P})$*, where*

- V *is a finite set of nodes, called the* state space*;*

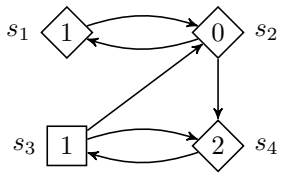
– $\rightarrow \subseteq V \times V$ *is a total transition relation;*
– $\Omega : V \to \mathbb{N}$ *is a function that assigns a* priority *to each node; and*
– $\mathcal{P} : V \to \{\diamond, \square\}$ *is a function that assigns a* player *to each node. The players are called* even ($\diamond$) *and* odd ($\square$).

We write $s \to t$ whenever $(s, t) \in \rightarrow$. The set of successors of a node s is denoted with $succ(s) = \{t \mid s \to t\}$. We use $\bigcirc$ to denote an arbitrary player and $\bar{\bigcirc}$ to denote its opponent.

A parity game is played as follows: initially, a token is placed on some node of the graph. The owner of the node can decide where to move the token; the token may be moved along one of the outgoing edges. This process continues ad infinitum, yielding an infinite path of nodes that the token moves through. Such a path is called a *play*. A play π is won by player $\diamond$ if the minimal priority that occurs infinitely often along π is even. Otherwise, it is won by player $\square$.

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

Example 2. Consider the parity game below. Here, priorities are inscribed in the nodes and the nodes are shaped according to their owner ($\diamond$ or $\square$).



In this game, the strategy $\sigma_\diamond$, partially defined as $\sigma_\diamond(s_1) = s_2$ and $\sigma_\diamond(s_2) = s_1$, is winning for $\diamond$ in s_1 and s_2. After all, the minimal priority that occurs infinitely often along $(s_1 s_2)^\omega$ is 0, which is even. Player $\square$ can win node s_3 with the strategy $\sigma_\square(s_3) = s_4$. Note that player $\diamond$ is always forced to move the token from node s_4 to s_3.

4 Partial-Order Reduction

A popular application of parity games is the encoding of model checking problems. Given a labelled transition system (LTS) and a formula in the modal mu-calculus, one can construct a parity game that expresses whether the formula holds for the LTS. In a typical translation, a (sub)formula holds for a given state if and only if the corresponding node in the parity game is won by player even.

However, in model checking, arbitrary interleaving of concurrent processes can lead to an exponential blowup in the size of the state space. Hence, most parity games that are constructed from those state spaces also grow rapidly in size. A popular technique to deal with this state-space explosion is *partial-order reduction* (POR). Several variants of POR exist, such as *ample sets* [18], *persistent sets* [7] and *stubborn sets* [20,21]. The current work is based on the stubborn set theory, since it does not depend on any knowledge about the underlying concurrent processes of a semantic model and it can deal with nondeterminism [22].

Since POR not only relies on node labels but also on edge labels, we assume the existence of some fixed set of edge labels $\mathcal{S}$, which we will call *events*. Events are typically denoted with the letter j. In Section 6, we will see where events originate, and how they relate to parity game transitions. We extend the standard definition of parity games to incorporate the edge labels:

Definition 2. *A* labelled parity game *is a directed graph* $G = (V, \rightarrow, \Omega, \mathcal{P})$*, where* V*,* Ω *and* $\mathcal{P}$ *are as before and* $\rightarrow \subseteq V \times \mathcal{S} \times V$ *is a total transition relation.*

We write $s \xrightarrow{j} t$ whenever $(s, j, t) \in \rightarrow$ and $s \rightarrow t$ when $s \xrightarrow{j} t$ for some j. The same notation is used to indicate the existence of longer paths $s \xrightarrow{j_1 \cdots j_n} t$. We say an event j is enabled in a node s, notation $s \xrightarrow{j}$, if and only if there is a transition $s \xrightarrow{j} t$ for some t. For a given parity game G, the set of all enabled events in a node s is denoted with $\mathit{enabled}_G(s)$. An event j is *invisible* if and only if for all transitions $s \xrightarrow{j} t$ labelled with j, it holds that $\mathcal{P}(s) = \mathcal{P}(t)$ and $\Omega(s) = \Omega(t)$. Otherwise, j is *visible*. From here on, we assume every parity game is labelled.

A central concept in POR is that of a *reduction function*, which indicates which edges to explore in each node. Given some initial node $\hat{s}$, a reduction function induces a unique *reduced parity game* as follows.

Definition 3. *Let* $G = (V, \rightarrow, \Omega, \mathcal{P})$ *be a parity game,* $\hat{s} \in V$ *a node and* $r : V \rightarrow 2^{\mathcal{S}}$ *a* reduction function*. Then the* reduced parity game *induced by* r *and starting from* $\hat{s}$ *is defined as* $G_r = (V_r, \rightarrow_r, \Omega, \mathcal{P})$*, where:*

- $\rightarrow_r$ *is the transition relation under* r*:* $\rightarrow_r = \rightarrow \cap \{(s, j, t) \mid j \in r(s)\}$*;*
- V_r *is the set of reachable nodes with* $\rightarrow_r$*:* $V_r = \{s \mid \hat{s} \rightarrow_r^* s\}$*, where* $\rightarrow_r^*$ *is the transitive closure of* $\rightarrow_r$*.*

Note that a reduced parity game induced by some reduction function r only has a total transition relation if $r(s) \cap \mathit{enabled}(s) \neq \emptyset$ for every reachable node s (this is enforced by the conditions we define below).

To really solve the issue of state-space explosion, one should not have to generate the full parity game. Instead, the reduction function should be computed either a priori [13] or *on-the-fly*, *i.e.*, during the exploration procedure. We take the latter approach.

In the general case, a reduction function is not guaranteed to preserve the winning player of a game. We identified the following properties from stubborn set theory which are required to preserve the winning player.

Definition 4. *Let $G = (V, \rightarrow, \Omega, \mathcal{P})$ be a parity game. A reduction function $r : V \rightarrow 2^S$ is a* weak stubborn set *iff for all nodes $s \in V$, the following conditions hold:*

D1 *For all $j \in r(s)$ and $j_1, \ldots, j_n \notin r(s)$, if $s \xrightarrow{j_1} s_1 \xrightarrow{j_2} \cdots \xrightarrow{j_n} s_n \xrightarrow{j} s'_n$, then there are nodes $s', s_1, \ldots, s'_{n-1}$ such that $s \xrightarrow{j} s' \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \cdots \xrightarrow{j_n} s'_n$ and $s_i \xrightarrow{j} s'_i$ for every $1 \leq i < n$.*

D2w *There is an event $j \in r(s)$ such that for all $j_1, \ldots, j_n \notin r(s)$, if $s \xrightarrow{j_1 \ldots j_n} s'$, then $s' \xrightarrow{j}$. Such an event is called a* key event*.*

V *If $r(s)$ contains an enabled visible event, then it contains all visible events.*

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

L *For every visible event j, every cycle in the reduced game contains a node s such that $j \in r(s)$.*

Below, we also use (weak) stubborn set to refer to the set of events $r(s)$ is some node s. First, note that every key event in a node s is enabled in s, by taking $n = 0$ in **D2w**. Condition **D1** ensures that whenever an enabled event is selected for the stubborn set, it should not disable other behaviour that is not in $r(s)$. Figure 1 shows a graphical representation of condition **D1**. A stubborn set can never be empty, due to **D2w**. In a traditional setting where POR is applied on an LTS, the combination of **D1** and **D2w** is sufficient to preserve deadlocks. Condition **V** enforces that either all visible events are selected for the stubborn set, or none are. Condition **L** prevents the so called *action-ignoring problem*, where a certain event is never selected for the stubborn set and ignored indefinitely. Since we assume that the state space is finite, it suffices to reason about the cycles of the reduced game. Finally, the combination of **I** and **L** helps to preserve divergences.

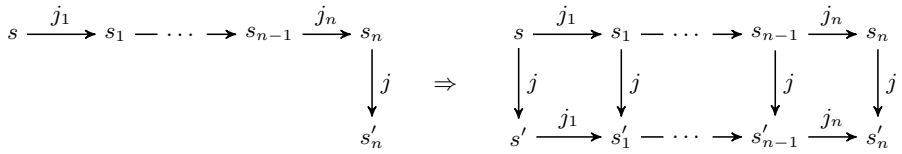
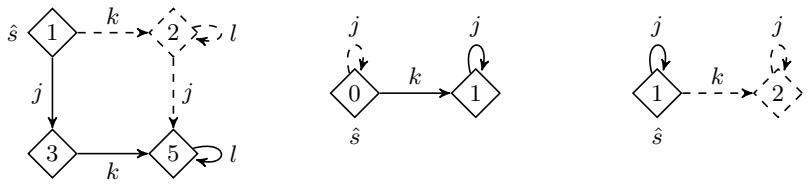


Fig. 1. Visual representation of condition **D1**.

Example 3. To further illustrate the purpose of conditions **V**, **I** and **L**, we give examples that show that the winning player in the original game and the reduced game might be different if one of these conditions is not satisfied. See Figure 2. From left to right, these games show a reduced game under a reduction function that does not satisfy **V**, **I** or **L**, respectively. In each case, we start exploration from the node called $\hat{s}$ and the winning strategy for player $\diamond$ in the original game is lost.

Note that the games in Figure 2 are from a subclass of parity games called *weak solitaire*. A game is considered *weak* when the priorities along all its paths



$\sigma_\diamond$: kl^ω — **D1**, **D2w**, **I**, **L**

$\sigma_\diamond$: j^ω — **D1**, **D2w**, **V**, **L**

$\sigma_\diamond$: kj^ω — **D1**, **D2w**, **V**, **I**

Fig. 2. Three games that show the winner is not necessarily preserved if we drop one of the conditions **V**, **I** or **L**, respectively. The dashed nodes and edges are present in the original game, but not in the reduced game. The edges taken from $\hat{s}$ by the winning strategy for player $\diamond$ in the original game are indicated below each game.

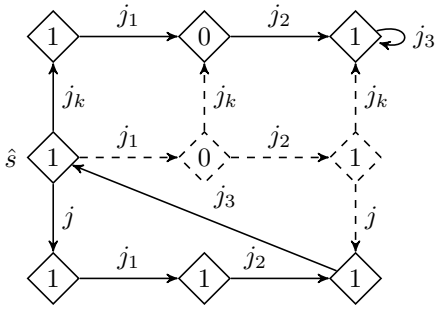
are non-decreasing, *i.e.*, if $s \to t$ then $\Omega(s) \leq \Omega(t)$. A game is a $\bigcirc$-*solitaire* game if all nodes belonging to player $\bar{\bigcirc}$ have exactly one successor. We call a game *solitaire* if it is $\bigcirc$-solitaire for some $\bigcirc \in \{\diamond, \Box\}$. Weak solitaire games can encode the model checking of safety properties. □

4.1 Strengthened condition D1

The condition **D1** that we propose is slightly stronger than the version that can be found in literature [14,20,23], which is as follows.

D1' For all $j \in r(s)$ and $j_1, \ldots, j_n \notin r(s)$, if $s \xrightarrow{j_1} \cdots \xrightarrow{j_n} s_n \xrightarrow{j} s'_n$, then there are nodes $s', s_1, \ldots, s'_{n-1}$ such that $s \xrightarrow{j} s' \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \cdots \xrightarrow{j_n} s'_n$.

However, this weaker condition may lead to the reduced game having a different winner than the original game, as shown by the following example. Consider the parity game below.



First, note that this game is deterministic. The events j_1 and j_2 are visible and j, j_k and j_3 are invisible. By setting $r(\hat{s}) = \{j, j_k\}$, which is valid under conditions **D1'**, **D2w**, **V**, **I** and **L**, the reduced game that starts from $\hat{s}$ does not contain the dashed nodes. The winning strategy for player $\diamond$ that moves through the node

with priority 0, via the edges labelled $j_1 j_2 j j_3$, is thus lost in the reduced game. A similar example can be constructed in the setting of POR for (deterministic) Kripke structures to show that LTL_{-X} is not necessarily preserved, contrary to what is suggested in [20,23].

We remark that this problem does not occur when applying *strong stubborn sets* [23] on a deterministic system, a more widely studied setting. In strong stubborn sets, **D2w** is replaced by condition **D2** below.

D2 For all events $j \in r(s)$ and $j_1, \ldots, j_n \notin r(s)$, if $s \xrightarrow{j_1 \ldots j_n} s'$, then $s' \xrightarrow{j}$.

In the deterministic case, conditions **D1'** and **D2** together imply **D1**.

4.2 Correctness

To show that the approach presented above is correct, we show in Theorem 1 that the winner is preserved in every node of the reduced game. We do this by constructing a strategy in the reduced game that mimics the strategy in the original game. The paths that are consistent with the two strategies must be *stutter equivalent* to preserve the winner.

Definition 5. *Let G be a parity game and $\pi = s_0 s_1 s_2 \ldots$ and $\pi' = t_0 t_1 t_2 \ldots$ be two paths in G. We say π and π' are stutter equivalent, notation $\pi \triangleq \pi'$, if and only if one of the following conditions holds:*

- *π and π' are both finite and there exists a non-decreasing partial function $f : \omega \to \omega$, with $f(0) = 0$ and $f(|\pi|-1) = |\pi'|-1$, such that for all $0 \leq i < |\pi|$ and $i' \in [f(i), f(i+1))$, it holds that $\mathcal{P}(s_i) = \mathcal{P}(t_{i'})$ and $\Omega(s_i) = \Omega(t_{i'})$.*
- *π and π' are both infinite and there exists an unbounded, non-decreasing total function $f : \omega \to \omega$, with $f(0) = 0$, such that for all i and $i' \in [f(i), f(i+1))$, it holds that $\mathcal{P}(s_i) = \mathcal{P}(t_{i'})$ and $\Omega(s_i) = \Omega(t_{i'})$.*

Lemma 1. *All infinite stutter equivalent paths have the same winner.*

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

In the lemmata and proofs below, we often use $\to_r$ to indicate which transitions must occur in the reduced state space.

Lemma 2. *Let r be a weak stubborn set and $\pi = s_0 \xrightarrow{j_1} \cdots \xrightarrow{j_n} s_n \xrightarrow{j} s'_n$ be a path such that $j_1, \ldots, j_n \notin r(s)$ and $j \in r(s)$. Then, there is a path $\pi' = s_0 \xrightarrow{j}_r s' \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$ such that π and π' are stutter equivalent.*

Proof. The existence of π' follows directly from condition **D1**. Due to condition **V** and our assumption that $j_1, \ldots, j_n \notin r(s_0)$, it cannot be the case that j is visible and at least one of $j_1, \ldots, j_n$ is visible. If j is invisible, then $s_0 \xrightarrow{j_1} \cdots \xrightarrow{j_n} s_n$ and $s'_0 \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$ have the same sequence of node labels, since **D1** implies that $s_i \xrightarrow{j} s'_i$ for every $0 \leq i \leq n$. Otherwise, if all of $j_1, \ldots, j_n$ are invisible, then the sequence of labels observed along π and π' has the shape $p^n q$ and $p q^n$, respectively, where p represents the labels of s_0 and q represents the labels of s'_0. We conclude that π and π' are stutter equivalent. □

Lemma 3. *Let r be a weak stubborn set and $\pi = s \xrightarrow{j_1} s_1 \xrightarrow{j_2} \ldots$ be a path such that $j_i \notin r(s)$ for any j_i that occurs in π. Then, the following holds:*

- *If π is of finite length n, there exists a node s'_n such that $s_n \xrightarrow{j_k} s'_n$ and a path $\pi' = s \xrightarrow{j_k}_r s' \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$.*
- *If π is infinite, there exists a path $\pi' = s \xrightarrow{j_k}_r s' \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \ldots$.*

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

Proof. Let K be the set of key events in s. If j_1 is invisible, K contains at least one invisible event, due to **I**. Otherwise, if j_1 is visible, we reason that K is not empty (condition **D2w**) and all events in $r(s)$, and thus also all events in K, are invisible, due to **V**. In the remainder, let j_k be an invisible key event.

In case π has finite length n, the existence of $s_n \xrightarrow{j_k} s'_n$ and $s \xrightarrow{j_k}_r s' \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$ follows from **D2w** and **D1**, respectively.

If π is infinite, we can apply **D2w** and **D1** successively to obtain a path $\pi_i = s \xrightarrow{j_k} s' \xrightarrow{j_1} \cdots \xrightarrow{j_i} s'_i$ for every $i \geq 0$. Since the state space is finite, at least one j_k-successor of s, which we refer to as t, must occur on infinitely many π_i. Thus, for every i, we have $t \xrightarrow{j_1 \ldots j_i}$, since t also occurs on some $\pi_{i'}$ with $i' > i$. We conclude the existence of $\pi' = s \xrightarrow{j_k}_r t \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \ldots$.

Since j_k is invisible, we use the same reasoning as in the proof of Lemma 2 to conclude $\pi \triangleq \pi'$. □

We remark that Lemma 3 also holds for parity games that have an infinite state space, but where all the invisible events are finitely branching. Lemma 3 thus shows that conditions **Ä1** and **Ä2** together imply **Ä3** (all from [22]) in LTSs that have finite branching of invisible actions.

The proof of the main theorem, Theorem 1, relies heavily on the constructions described by Lemma 2 and 3. The following example provides insight into the application of the lemmata.

Example 4. Consider the path $j_1 j_2 j_3$ in Figure 3. This path is mimicked by the path $j_k j_2 j_1 j'_k j_3$, drawn with dashes. The new path reorders the events j_1, j_2 and j_3 according to the construction of Lemma 2 and introduces the key events j_k and j'_k according to the construction of Lemma 3.

We need one additional lemma to show that the initial event of a path will be selected for the stubborn set if Lemma 2 cannot be applied to that path.

Lemma 4. *Let r be a weak stubborn set and $s \xrightarrow{j_1} \xrightarrow{j_2} \cdots \xrightarrow{j_n} s_n$ be a path such that $j_2, \ldots, j_n$ are disabled in s and $j_n \in r(s)$. Then, it must be the case that $j_1 \in r(s)$.*

Proof. By induction. The base case, where $n = 1$, trivially satisfies the condition, since j_n coincides with j_1. For the inductive case, we assume as induction hypothesis that for all paths $s \xrightarrow{j_1} \cdots \xrightarrow{j_l} s_l$ with $l \leq i$, it holds that if $j_2, \ldots, j_l$ are disabled in s and $j_l \in r(s)$, then it holds that $j_1 \in r(s)$. Let $s \xrightarrow{j_1} \cdots \xrightarrow{j_i} s_i \xrightarrow{j_{i+1}} s_{i+1}$ be some path of length $i + 1$ that fulfils these same

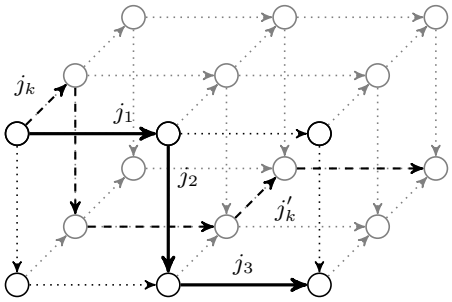


Fig. 3. Example of how the trace j_1, j_2, j_3 can be mimicked (dashed trace) by introducing additional events and moving j_2 to the front. Transitions that are drawn in parallel have the same label.

conditions, *i.e.*, $j_2, \ldots, j_i, j_{i+1}$ are disabled in s and $j_{i+1} \in r(s)$. Since the path $s \xrightarrow{j_{i+1} j_1 \cdots j_i}$ is not enabled, condition **D1** can only be fulfilled by setting $j_l \in r(s)$ for some $l \leq i$. Consequently, we obtain a path $s \xrightarrow{j_1} \cdots \xrightarrow{j_l}$, where $j_2, \ldots, j_l$ are disabled in s and $j_l \in r(s)$. Applying the induction hypothesis to this path yields $j_1 \in r(s)$. □

The following theorem shows that partial-order reduction preserves the winning player in all nodes of the reduced game. Its proof is inspired by [20] and [2, Lemma 8.21].

Theorem 1. *Let G be a parity game, r a weak stubborn set and G_r the reduced game according to r starting from some node $\hat{s}$. If G_r is finite, then it holds that for every node s in G_r, the winner of s in G_r is equal to the winner of s in G.*

Proof. Let G be a parity game and G_r a finite reduced game induced by some reduction function r that satisfies conditions **D1**, **D2w**, **V**, **I** and **L**. Let player $\bigcirc$ be the winner of some node s.

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

Otherwise, if $\mathcal{P}(s) = \bigcirc$, let σ be some winning strategy for $\bigcirc$ in s under $\rightarrow$. To prove that $\bigcirc$ also wins node s in G_r, we construct a matching strategy σ' that $\bigcirc$ should follow in G_r. By showing that for every path π in G_r that is consistent with σ', a stutter equivalent path π' can occur in G when following the original strategy σ, we prove that σ' is indeed a winning strategy for $\bigcirc$ starting from s in G_r.

Consider the path $\pi_0 = s \xrightarrow{k_1} s_1 \xrightarrow{k_2} \ldots$ that is generated by player $\bigcirc$ moving the token according to σ. The path π_0 is finite if and only if player $\bigcirc$ at some point moves the token to a node owned by player $\bar{\bigcirc}$. We will construct a corresponding path $\hat{\pi}_0$ in the reduced state space. Then, we define σ' such that player $\bigcirc$ moves the token along $\hat{\pi}_0$. Note that although this only defines σ' on a

part of the game, the same reasoning can be applied to all other nodes belonging to player $\bigcirc$.

From π_0, we will step-by-step construct new paths π_i that are stutter equivalent to π_0, by shifting events forward (construction of Lemma 2) and introducing key events (construction of Lemma 3). Since the first step of σ can be trivially imitated if $k_1 \in r(s)$, we henceforth assume that $k_1 \notin r(s)$. Each path π_i is thus of the shape

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

where $j_1, \dots, j_i$ are key events in the stubborn set. These can either originate from $k_2, k_3, \dots$, *i.e.*, events that are shifted forward with the construction from Lemma 2, or they can be events that are newly introduced using the construction from Lemma 3. The events $l_1^i, l_2^i \dots$ represent the remaining subsequence of $k_2, k_3 \dots$, *i.e.*, those events that have not been shifted forward (yet). Note that $j_1, \dots, j_i$ can only contain a visible event if the rest of π_i is invisible (cf. the proofs of Lemma 2 and Lemma 3).

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

- None of the events $k_1, l_1^i, l_2^i, \dots$ is visible. In that case, player $\bigcirc$ never moves the token to a node belonging to player $\bar{\bigcirc}$ and the path π_0 is infinite. If k_1 is never taken, we can mimic the divergent behaviour of π_i by applying Lemma 3 ad infinitum on state $t_i, t_{i+1}, \dots$.
- There is a visible event $m \in \{k_1, l_1^i, l_2^i, \dots\}$. We consider a π_i such that all events that fulfil the requirements of Lemma 2 have already been shifted forward, *i.e.*, none of $l_0^i, l_1^i, \dots$ is enabled in t_i. This path exists due to finiteness of $\rightarrow_r$. Since the reduced game is finite and the event m is selected at least once on every cycle in the reduced game (condition **L**), there is a $\pi_{i'}$ with $i' \geq i$ such that $m \in r(t_{i'})$. None of the events $l_0^i, l_1^i, \dots$ is enabled in $t_{i'}$ (they did not satisfy Lemma 2, after all), therefore – by Lemma 4 – it must be the case that $k_1 \in r(t_{i'})$.

In either case, as i goes to ω, we obtain a path $\hat{\pi}_0$. This path is stutter equivalent to π_0, since each π_i is stutter equivalent to its predecessor.

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

In case $\hat{\pi}_0$ ends in a node owned by $\bar{\bigcirc}$, π_0 is also finite and we reason as follows. From its definition, we know that the last node of π_0, denoted s_n, is owned by player $\bar{\bigcirc}$. There is a path in the original state space from s_n to the last node of $\hat{\pi}_0$, denoted $\hat{s}$, namely along those events of $\hat{\pi}_0$ that were introduced by Lemma 3. We call this path π_{key}. We obtain a path $\pi_0\pi_{key}$, which is π_0 extended with invisible key events introduced by Lemma 3 in $\hat{\pi}_0$. Since π_{key} contains only invisible events, all nodes on π_{key} are owned by $\bar{\bigcirc}$ and $\pi_0\pi_{key}$ is consistent with σ. Extending a path with finitely many invisible events is

permitted under stutter equivalence, hence we conclude that $\hat{\pi}_0$ and $\pi_0\pi_{key}$ are stutter equivalent. □

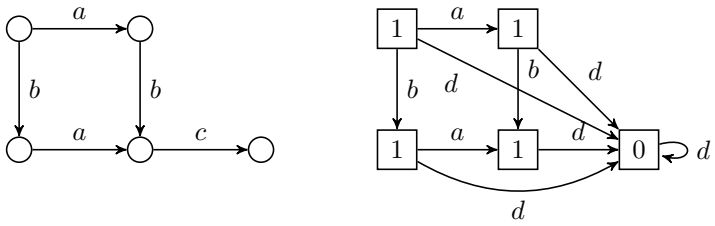
5 Weakening the Conditions

The theory we have introduced above depends heavily on rectangular structures in the parity game. This is especially apparent in condition **D1**. However, parity games obtained from model checking problems also often contain triangular structures, as the following example demonstrates.

Example 5. Consider the process $(a \parallel b) \cdot c$ with the mu-calculus property

$$\mu X.(\bigwedge_{\alpha\in\{a,b\}} [\alpha]X \wedge \bigvee_{\alpha\in\{a,b,c\}} \langle\alpha\rangle true)$$

which expresses that the action c must unavoidably be done within a finite number of steps. Below, the LTS is depicted on the left and a possible parity game encoding of our liveness property on this state space is depicted on the right. The edges in the parity game that originate from the subformula $\langle true\rangle true$ are labelled with d.



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

In order to deal with games that contain triangular structures, we need to weaken condition **D2w**.

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

Most of the correctness proofs from Section 4.2 still apply, we only have to alter Lemma 3 to deal with the updated condition **D2t**.

Lemma 5. *Let r be a weak stubborn set and $\pi = s \xrightarrow{j_1} s_1 \xrightarrow{j_2} \ldots$ be a path such that $j_i \notin r(s)$ for any j_i that occurs in π. Then, the following holds:*

- *If π is of finite length n, there exists a node s'_n and a path π' such that:*

- $s_n \xrightarrow{j_k} s'_n$ *or* $s_n = s'_n$*; and*
- $\pi' = s \xrightarrow{j_k}_r s' \xrightarrow{j_1} \cdots \xrightarrow{j_n} s'_n$

– *If* π *is infinite, there exists a path* $\pi' = s \xrightarrow{j_k}_r s' \xrightarrow{j_1} s'_1 \xrightarrow{j_2} \ldots$.

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

Proof. We follow the same reasoning as in the proof of Lemma 3 to conclude the existence of an invisible key event j_k.

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

If π is infinite, we distinguish the following cases:

– If $s \xrightarrow{j_k j_1 \ldots j_i} s_i$ for some i, we can trivially extend this path to obtain $\pi' = s \xrightarrow{j_k j_1 \ldots j_i} s_i \xrightarrow{j_{i+1}} \ldots$.
– Otherwise, if there is no i such that $s \xrightarrow{j_k j_1 \ldots j_i} s_i$, we can apply the same reasoning as in the proof of Lemma 3.

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

We remark that these ideas are similar to the more general concept of weak confluence from [8]. However, weak confluence is very hard to compute, so we decided to work with a more restricted condition.

6 Parameterised Boolean Equation Systems

The previous sections explained how parity games can be reduced under POR. In a practical setting, this reduction should be applied while generating the parity game. In the following sections, we explain how this can be achieved using *parameterised Boolean equation systems*, which can compactly represent parity games.

In this section, we rely on abstract data types and their non-empty data sorts, which are denoted with the letters D and E. The corresponding semantic domains are $\mathbb{D}$ and $\mathbb{E}$. Furthermore, B and N represent the Booleans and the natural numbers respectively, and have $\mathbb{B}$ and $\mathbb{N}$ as semantic counterpart. The set of data variables is $\mathcal{V}$, and its elements are usually denoted with d and e. To interpret expression with variables, we use a *data environment* δ, which maps every variable in $\mathcal{V}$ to an element of the corresponding sort. The semantics of an expression f in the context of an environment δ is denoted $[\![f]\!]\delta$. To update an environment, we use the notation $\delta[v/d]$, which is defined as $\delta[v/d](d) = v$ and $\delta[v/d](d') = \delta(d')$ for all variables $d \neq d'$.

Here, we restrict ourselves to giving a short introduction to PBESs; the interested reader is referred to [9].

Definition 6. *A* predicate formula *is defined by the following grammar:*

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

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

$$
\begin{array}{rcl@{\qquad}rcl}
[\![b]\!]\eta\delta & \Leftrightarrow & [\![b]\!]\delta & [\![X(f)]\!]\eta\delta & \Leftrightarrow & [\![f]\!]\delta \in \eta(X) \\
[\![\varphi \wedge \psi]\!]\eta\delta & \Leftrightarrow & [\![\varphi]\!]\eta\delta \textit{ and } [\![\psi]\!]\eta\delta \textit{ hold} & [\![\varphi \vee \psi]\!]\eta\delta & \Leftrightarrow & [\![\varphi]\!]\eta\delta \textit{ or } [\![\psi]\!]\eta\delta \textit{ hold} \\
[\![\varphi \Rightarrow \psi]\!]\eta\delta & \Leftrightarrow & [\![\varphi]\!]\eta\delta \textit{ holds implies that } [\![\psi]\!]\eta\delta \textit{ holds} & & & \\
[\![\forall d\colon E.\ \varphi]\!]\eta\delta & \Leftrightarrow & \textit{for all } v \in \mathbb{E},\ [\![\varphi]\!]\eta\delta[v/d] \textit{ holds} & & & \\
[\![\exists d\colon E.\ \varphi]\!]\eta\delta & \Leftrightarrow & \textit{for some } v \in \mathbb{E},\ [\![\varphi]\!]\eta\delta[v/d] \textit{ holds} & & &
\end{array}
$$

A predicate formula is *syntactically monotone* iff no predicate variable occurs on the left-hand side of an implication. Without loss of generality, in this paper we only consider PBESs where all the predicate variables have exactly one parameter of the same data sort D. In the examples, we may use predicate variables with multiple parameters.

Definition 7. *A* parameterised Boolean equation system *(PBES) is a sequence of equations that follow the grammar*

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

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

In the remainder, we often omit the trailing $\emptyset$. The right-hand side of an equation for X is called φ_X. The set of predicate variables that are bound in $\mathcal{E}$, *i.e.*, those that occur on the left-hand side of an equation, is denoted with $\mathsf{bnd}(\mathcal{E})$. Furthermore, the *signature* of a PBES $\mathcal{E}$ is defined as $\mathsf{sig}(\mathcal{E}) = \mathsf{bnd}(\mathcal{E}) \times \mathbb{D}$. Every predicate variable bound in $\mathcal{E} = (\sigma_1 X_1(d{:}D) = \varphi_1) \ldots (\sigma_n X_n(d{:}D) = \varphi_n)$ is assigned a *rank*, where $\mathsf{rank}_{\mathcal{E}}(X_i)$ is the number of alternations in the sequence of fixpoint symbols $\nu\sigma_1\sigma_2 \ldots \sigma_i$. Observe that $\mathsf{rank}_{\mathcal{E}}(X_i)$ is *even* iff $\sigma_i = \nu$. A PBES is *closed* if and only if all data variables occurring in a right-hand side φ_X are either bound in a quantifier or as a parameter of X and all predicate variables in φ_X are in $\mathsf{bnd}(\mathcal{E})$. We say a PBES is *well-formed* iff there is exactly one equation for every predicate variable $X \in \mathsf{bnd}(\mathcal{E})$. Henceforth, we assume all PBESs are closed and well-formed.

Definition 8. *The* solution *$[\![\mathcal{E}]\!]\eta\delta$ of a PBES $\mathcal{E}$ in the context of a predicate environment η and a data environment δ, is a predicate environment that is defined inductively:*

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

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

The intuition behind the solution of a PBES is that priority is given to the fixed points of equations that occur early in the PBES, while the equalities specified by the equations are always satisfied. The existence of the least fixpoint μT_X and the greatest fixpoint νT_X in the complete lattice $(2^{\mathbb{D}}, \subseteq)$ is ensured by the monotonicity of the transformer $T_X : 2^{\mathbb{D}} \to 2^{\mathbb{D}}$, which follows from the syntactic monotonicity of φ_X. Since the solution of the bound variables of a closed PBES does not depend on the environments η and δ, we often write $[\![\mathcal{E}]\!]$ instead of $[\![\mathcal{E}]\!]\eta\delta$.

We use a normal form called *standard recursive form* (SRF) [17] to facilitate reasoning symbolically about the dependencies between predicate variables.

Definition 9. *Let $\mathcal{E}$ be a PBES. Then $\mathcal{E}$ is in* standard recursive form *(SRF) iff for all $(\sigma_i X_i(d{:}D) = \phi) \in \mathcal{E}$, ϕ is either disjunctive or conjunctive,* i.e.*, the equation for X_i has the shape*

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

or

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

Furthermore, we add the semantic restriction that for every $(X, v) \in \mathsf{sig}(\mathcal{E})$, at least one condition f_j should evaluate to true, i.e.*, there is a $j \in J$, a data environment δ and a $v_j \in \mathbb{E}_j$ such that $[\![f_j(d, e_j)]\!]\delta[v_j/e_j, v/d]$ holds.*

Every clause in the right-hand side of the equation for X_i corresponds to a unique event (cf. Section 3). Henceforth, we assume that the index sets for events are disjunct for different equations, *i.e.*, given predicate variables X_1, X_2, then $J_1 \cap J_2 = \emptyset$. This allows us to uniquely identify each event by its index $j \in J_i$. The set of all events in a PBES $\mathcal{E}$ is denoted with $\mathsf{evt}(\mathcal{E})$, defined as $\mathsf{evt}(\mathcal{E}) = \bigcup_{X_i \in \mathsf{bnd}(\mathcal{E})} J_i$. We use the function $\mathsf{op}_{\mathcal{E}} : \mathsf{bnd}(\mathcal{E}) \to \{\vee, \wedge\}$ to indicate for each predicate variable in a PBES in SRF whether the associated equation is disjunctive or conjunctive. We say an event $j \in J_i$ is *invisible* if the rank and operand are not affected by j, in other words, if $\mathsf{rank}_{\mathcal{E}}(X_i) = \mathsf{rank}_{\mathcal{E}}(X_j)$ and $\mathsf{op}_{\mathcal{E}}(X_i) = \mathsf{op}_{\mathcal{E}}(X_j)$. Otherwise, it is *visible*.

Definition 10. *Let $\mathcal{E}$ be a PBES in SRF, where each equation has the same structure as in Definition 9. Then, the* parity game *of $\mathcal{E}$ is defined as $G = (\mathsf{sig}(\mathcal{E}), \to, \Omega, \mathcal{P})$, where*

- *$\to$ is the transition relation, which satisfies $(X_i, v) \xrightarrow{j} (X_j, w)$ for given X_i, $j \in J_i$, v and w if and only if for some δ, both $[\![f_j(d, e_j)]\!]\delta[v/d]$ and $w = [\![g_j(d, e_j)]\!]\delta[v/d]$ hold;*
- *$\Omega((X, v)) = \mathsf{rank}_{\mathcal{E}}(X)$; and*
- *$\mathcal{P}((X, v)) = \Diamond$ iff $\mathsf{op}_{\mathcal{E}}(X) = \vee$.*

We remark that parity games constructed according to the above definition often have an infinite state space, *e.g.*, when the PBES has a parameter that is a natural number. In practice, we only consider the part of the parity game that is reachable from some initial node (X, v).

The following theorem is adapted from [5] and [17], in which parity games and winning strategies are called *dependency space* [17] and *proof graph* [5], respectively.

Theorem 2 ([5]). *Let $\mathcal{E}$ be a PBES with $X \in \mathsf{bnd}(\mathcal{E})$. Then $v \in [\![\mathcal{E}]\!](X)$ iff there is a winning strategy for player $\Diamond$ from (X, v). Dually, $v \notin [\![\mathcal{E}]\!](X)$ iff there is a winning strategy for player $\Box$ from (X, v).*

Example 6. Consider the PBES

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

The six nodes in the parity game which are reachable from (X, true) are depicted in Figure 4. The horizontally drawn edges all stem from the clause $\exists n{:}N.\, n \leq 2 \wedge Y(b, \mathit{if}(b, n, 0))$, and are thus labelled with the same event (not shown in the figure). Vertical edges are labelled with $b \wedge X(\mathit{false})$ (on the left) or with $Y(\mathit{false}, 0)$ (on the right). The selfloop is also labelled with $Y(\mathit{false}, 0)$. All nodes in this game are won by player $\Box$, and thus $\mathit{true} \notin [\![\mathcal{E}]\!](X)$ according to Theorem 2. □

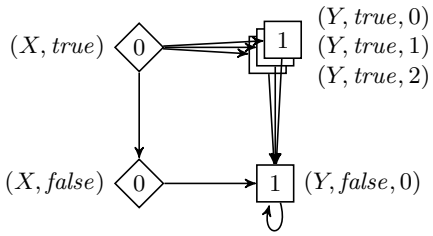


Fig. 4. Reachable part of the parity game underlying the PBES of Example 6, when starting from node (X, true).

7 Implementation

In this section, we give details of a possible implementation of POR on PBESs. This implementation partially implements the framework of [14] and extends it with non-determinism.

Conditions **D1**, **D2w** and **L** are properties of the (reduced) state space as a whole and they are hard to check locally. Therefore, we need alternative stronger

conditions that can help to construct a stubborn set *on-the-fly*. The most common local condition for $\mathbf{L}$ is the so called *stack proviso* $\mathbf{L}^S$ [18]. This proviso assumes that the state space is explored with *depth-first search* (DFS), and it uses the *Stack* that stores unexplored nodes to determine whether a cycle is being closed. If so, the node will be *fully expanded*, *i.e.*, $r(s) = enabled_G(s)$.

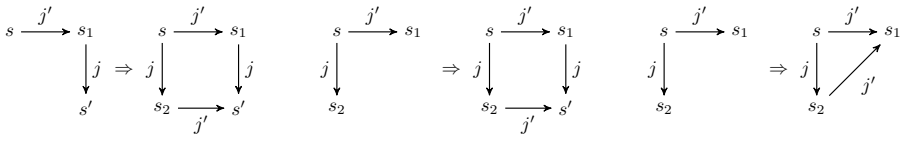
$\mathbf{L}^S$ For all nodes $s \in V_r$, either $succ_{G_r}(s) \cap Stack = \emptyset$ or $r(s) = enabled_G(s)$.

To decide locally whether **D1** and **D2w** are satisfied, we need to perform several static analyses on the PBES. To reason about which events are in some sense independent, we rely on the idea of *accordance*. We define four flavours of accordance, which are all binary relations on events: *DNL*, *DNS*, *DNT* and *DNA*.

Definition 11. *Let $\mathcal{E}$ be a PBES in SRF and $G = (V, \rightarrow, \Omega, \mathcal{P})$ its parity game. Then, we define the following* accordance *relations:*

- *An event j* left-accords *with an event j' if it holds that for all nodes s, if $s \xrightarrow{j'} s_1 \xrightarrow{j} s'$, then $s \xrightarrow{j} s_2 \xrightarrow{j'} s'$ for some node s_2. If j does not left-accord with j', we write $(j, j') \in DNL$.*
- *Two events j and j'* square-accord *if it holds that for all nodes s, if $s \xrightarrow{j} s_1$ and $s \xrightarrow{j'} s_2$, then $s_1 \xrightarrow{j'} s'$ and $s_2 \xrightarrow{j} s'$ for some node s'. If two events j and j' do not accord, we write $(j, j') \in DNS$.*
- *An event j* triangle-accords *with j' if it holds for all nodes s, if $s \xrightarrow{j'} s_1$ and $s \xrightarrow{j} s_2$, then $s_2 \xrightarrow{j'} s_1$. If j does not triangle-accord with j', then we write $(j, j') \in DNT$.*
- *An event j* accords *with j' if j and j' square-accord or if j triangle-accords with j'. If j does not accord with j', we write $(j, j') \in DNA$.*

Given a relation R on events, we use $R(j)$ to denote its left projection: $R(j) = \{j' \mid (j, j') \in R\}$. The definition of left-according, square-according and triangle-according respectively can be graphically represented as follows.



Note that the relations *DNL* and *DNT* are not necessarily symmetric.

Apart from computing an accordance relation between events, which may be over-approximated, we also need to compute in which way events might become enabled. For this, we use *necessary enabling sets* [7].

Definition 12. *Let j be an event that is disabled in some node s. A* necessary-enabling set *(NES) for j in s is a set of events $NES_s(j)$ such that for all paths $s \xrightarrow{j_1 \cdots j_n J}$, there is at least one i such that $j_i \in NES_s(j)$.*

For every node and event there might be more than one NES. In particular, every superset of a NES is also a NES. A simple approach to calculate a NES, in a PBES with multiple parameters per predicate variable, is investigating which parameters influence the validity of conditions f_j and which parameters are changed in the update functions g_j. More accurate results can be achieved with advanced techniques to extract the control flow from a PBES [11]. Over-approximating a NES does not influence the correctness of this approach.

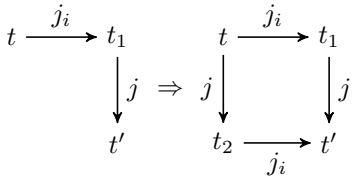
The following three lemmata show how the accordance relations and necessary-enabling set can be used to implement conditions **D1**, **D2w** and **D2t**, respectively. A combination of Lemma 6 and 7 in a deterministic setting appeared as Lemma 1 in [14]. Non-determinism does not affect the proof of Lemma 7, so the exact same reasoning can also be found in [14].

Lemma 6. *A reduction function r satisfies condition **D1** in a node s if for all $j \in r(s)$:*

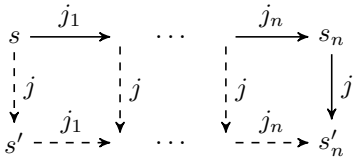
- *if j is disabled in s, then $NES_s(j) \subseteq r(s)$ for some NES_s; and*
- *if j is enabled in s, then $DNL(j) \subseteq r(s)$.*

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

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



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

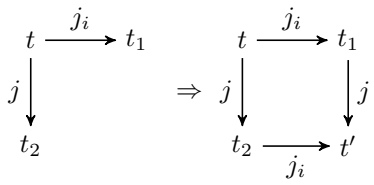


We conclude that the conditions of **D1** are satisfied.

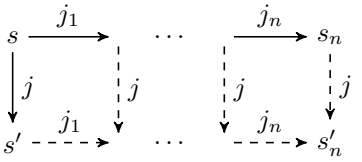
□

Lemma 7. *A reduction function r satisfies condition **D2w** in a node s if there is an enabled event $j \in r(s)$ such that $DNS(j) \subseteq r(s)$.*

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



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

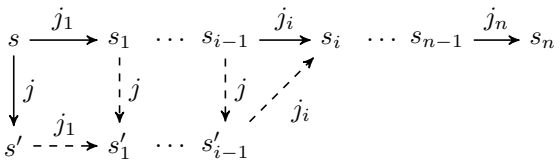


Hence, j satisfies the conditions of **D2w**. □

Lemma 8. *A reduction function r satisfies condition **D2t** in a node s if there is an enabled event $j \in r(s)$ such that $DNA(j) \subseteq r(s)$.*

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

- It holds that $j_1, \dots, j_n \in DNT(j)$. Since $j_1, \dots, j_n \notin r(s)$ and $r(s) \supseteq DNA(j) = DNS(j) \cap DNT(j)$, we can deduce that $j_1, \dots, j_n \notin DNS(j)$. By following the same reasoning as in the proof of Lemma 7, we derive the validity of **D2t**.
- There is an $0 < i \leq n$ such that $j_i \notin DNT(j)$. We consider the smallest such i, *i.e.*, $j_1, \dots, j_{i-1} \in DNT(j)$. With $j_1, \dots, j_n \notin r(s)$ and $r(s) \supseteq DNA(j) = DNS(j) \cap DNT(j)$, we deduce that $j_1, \dots, j_{i-1} \notin DNS(j)$ and $j_i \notin DNT(j)$. By first applying the square-according relation from left to right on j and $j_1, \dots, j_{i-1}$ and then applying the triangle-according relation on j and j_i, we derive the existence of the dashed transitions in the following figure.



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

□

7.1 Identifying Similar Events

In PBESs, the same transitions may be encoded in multiple equations. To identify the events that encode a similar transition, we rely on the notion of *event equivalence*.

Definition 13. *Let $\mathcal{E}$ be a PBES in SRF and $X_1, X_2 \in \mathsf{bnd}(\mathcal{E})$. Then the relation $\sim \subseteq \mathsf{evt}(\mathcal{E}) \times \mathsf{evt}(\mathcal{E})$ is such that given two events $j \in J_1$ and $j' \in J_2$, it holds that $j \sim j'$ if and only if for all $v \in \mathbb{D}$:*

$$\{v' \mid \exists \delta.\, [\![f_j(d, e_j)]\!]\delta[v/d] \wedge v' = [\![g_j(d, e_j)]\!]\delta[v/d]\} =$$
$$\{v' \mid \exists \delta.\, [\![f_{j'}(d, e_{j'})]\!]\delta[v/d] \wedge v' = [\![g_{j'}(d, e_{j'})]\!]\delta[v/d]\}$$

and j and j' are both visible or both invisible.

The equivalence relation $\sim$ partitions $\mathsf{evt}(\mathcal{E})$ into equivalence classes. By labelling the parity game of $\mathcal{E}$ with these equivalence classes, more reduction can be achieved in some cases.

7.2 Deterministic Events

So far, we have assumed that the PBESs we consider are non-deterministic. This forces us to use the left-accordance relation to satisfy **D1** (cf. the proof of Lemma 6). More reduction can be achieved if a PBES is partly or completely deterministic and some of the conditions can be relaxed. To this end, we identify the concept of a *deterministic event*: an event j is deterministic if for all nodes t, t' and t'', if $t \xrightarrow{j} t'$ and $t \xrightarrow{j} t''$, then it must be that $t' = t''$. We can statically compute which events are deterministic (not considering reachability) with the following function:

$$det(j) = [\![\forall d, e_j, e'_j.\, (f_j(d, e_j) \wedge f_j(d, e'_j)) \Rightarrow g_j(d, e_j) = g_j(d, e'_j)]\!]\delta$$

where δ is an arbitrary data environment.

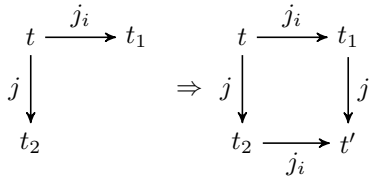
The following lemma expands on Lemma 6 and shows how knowledge of deterministic summands can be applied to potentially improve the reduction.

Lemma 9. *A reduction function r satisfies condition **D1** in a node s if for all $j \in r(s)$:*

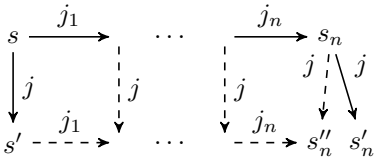
– *if j is disabled in s, then $NES_s(j) \subseteq r(s)$ for some NES_s; and*
– *if j is deterministic and enabled in s, then $DNS(j) \subseteq r(s)$ or $DNL(j) \subseteq r(s)$.*
– *if j is non-deterministic and enabled in s, then $DNL(j) \subseteq r(s)$.*

Proof. Let s be an arbitrary node and let r be a reduction function that satisfies the conditions above. For the cases where j is disabled or j is enabled and $DNL(j) \subseteq r(s)$, see the proof of Lemma 6. Here, we only consider the new case where j is deterministic and enabled in s and $DNS(j) \subseteq r(s)$.

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



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



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

The sets *DNS* and *DNL* are incomparable, so we cannot decide a priori which should be used for deterministic transitions. However, Lemma 9 permits choosing one of the accordance sets on-the-fly. This choice can be made based on a heuristic function, similar to the function for NESs proposed in [14].

8 Experiments

We implemented the ideas from the previous section in a prototype tool built on the mCRL2 toolset [4]. We performed a first experiment with a modified version of Milner's Scheduler [16], that includes, for each of the components, the possibility to break down with the action *disaster*. Whereas the LTS of the standard version of Milner's Scheduler can be greatly reduced by applying τ-confluence, this is not possible on our adapted example. Hence, the state space grows quickly with the number of components. In this experiment, we check the property "as long as no disaster occurs, there is no deadlock". We construct a PBES for several instances of the model, each with a different number of

Table 1. Results of applying POR to Milner's Scheduler with breakdowns. The size of the original and the reduced state space are denoted by $|V|$ and $|V_r|$, respectively.

Number of components	2	3	4	5	6	8	10	12
$\|V\|$	13	37	97	241	577	3073	15361	73729
$\|V_r\|$	10	14	18	22	26	34	42	50

components. From this PBES, we generate both the original parity game and the reduced parity game. The results are listed in Table 1.

As can be seen on the first row of the table, the size of the original state space grows quickly. However, the reduced state space only grows with four nodes per added component. It is thus feasible to check much larger instances of this model when applying POR on parity games.

9 Conclusion

We have presented an approach for applying partial-order reduction on parity games. This has two main advantages over POR applied on LTSs or Kripke structures: our approach supports the full modal mu-calculus, not just a fragment thereof, and the potential for reduction is greater, because we do not require a singleton proviso. Furthermore, we have shown how the ideas can be implemented with PBESs as a high-level representation. In future work, we aim to improve the prototype implementation and perform a complete experimental evaluation to validate its effectiveness. We also want to investigate the possibility of solving a reduced parity game while is it being constructed. In certain cases, one may be able to decide the winner of the original game from this partial solution.

Acknowledgements The authors would like to thank Antti Valmari for his discussions on the correctness of **D1'** and for providing the counter-example of Section 4.1.

References

1. Baier, C., D'Argenio, P., Groesser, M.: Partial Order Reduction for Probabilistic Branching Time. Electronic Notes in Theoretical Computer Science **153**(2 SPEC. ISS.), 97–116 (2006). https://doi.org/10.1016/j.entcs.2005.10.034
2. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
3. Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: CONCUR 1998. LNCS, vol. 1466, pp. 485–500 (1998). https://doi.org/10.1007/bfb0055643
4. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, J.W., Wijs, A.W., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems: Improvements in Expressivity and Usability. In: TACAS 2019. LNCS, vol. 11428, pp. 21–39 (2019). https://doi.org/10.1007/978-3-030-17465-1_2

5. Cranen, S., Luttik, B., Willemse, T.A.C.: Proof graphs for parameterised Boolean equation systems. In: CONCUR 2013. LNCS, vol. 8052, pp. 470–484 (2013). https://doi.org/10.1007/978-3-642-40184-8_33
6. Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A Partial Order Approach to Branching Time Logic Model Checking. Information and Computation **150**(2), 132–152 (1999). https://doi.org/10.1006/inco.1998.2778
7. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems, LNCS, vol. 1032. Springer (1996). https://doi.org/10.1007/3-540-60761-7
8. Groote, J.F., Sellink, M.P.A.: Confluence for process verification. Theoretical Computer Science **170**(1-2), 47–81 (1996). https://doi.org/10.1016/s0304-3975(96)00175-2
9. Groote, J.F., Willemse, T.A.C.: Parameterised boolean equation systems. Theoretical Computer Science **343**(3), 332–369 (2005). https://doi.org/10.1016/j.tcs.2005.06.016
10. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods in System Design **9**(1-2), 41–75 (1996). https://doi.org/10.1007/BF00625968
11. Keiren, J.J.A., Wesselink, J.W., Willemse, T.A.C.: Liveness Analysis for Parameterised Boolean Equation Systems. In: ATVA 2014. LNCS, vol. 8837, pp. 219–234 (2014). https://doi.org/10.1007/978-3-319-11936-6_16
12. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science **27**(3), 333–354 (1982). https://doi.org/10.1007/BFb0012782
13. Kurshan, R., Levin, V., Minea, M., Peled, D., Yenigün, H.: Static partial order reduction. In: TACAS 1998. LNCS, vol. 1384, pp. 345–357 (1998). https://doi.org/10.1007/BFb0054182
14. Laarman, A., Pater, E., van de Pol, J., Hansen, H.: Guard-based partial-order reduction. STTT **18**(4), 427–448 (2016). https://doi.org/10.1007/s10009-014-0363-9
15. Liebke, T., Wolf, K.: Taking Some Burden Off an Explicit CTL Model Checker. In: PETRI NETS 2019. LNCS, vol. 11522, pp. 321–341 (2019). https://doi.org/10.1007/978-3-030-21571-2_18
16. Milner, R.: A Calculus of Communicating Systems, LNCS, vol. 92. Springer (1980)
17. Neele, T., Willemse, T.A.C., Groote, J.F.: Solving Parameterised Boolean Equation Systems with Infinite Data Through Quotienting. In: FACS 2018. LNCS, vol. 11222, pp. 216–236 (2018). https://doi.org/10.1007/978-3-030-02146-7_11
18. Peled, D.: All from One, One for All: on Model Checking Using Representatives. In: CAV 1993. LNCS, vol. 697, pp. 409–423 (1993). https://doi.org/10.1007/3-540-56922-7_34
19. Ramakrishna, Y.S., Smolka, S.A.: Partial-Order Reduction in the Weak Modal Mu-Calculus. In: CONCUR 1997. LNCS, vol. 1243, pp. 5–24 (1997). https://doi.org/10.1007/3-540-63141-0_2
20. Valmari, A.: A Stubborn Attack on State Explosion. Formal Methods in System Design **1**(4), 297–322 (1992). https://doi.org/10.1007/BF00709154
21. Valmari, A.: The state explosion problem. In: ACPN 1996. LNCS, vol. 1491, pp. 429–528 (1996). https://doi.org/10.1007/3-540-65306-6_21
22. Valmari, A.: Stubborn Set Methods for Process Algebras. In: POMIV 1996. DIMACS, vol. 29, pp. 213–231 (1997). https://doi.org/10.1090/dimacs/029/12
23. Valmari, A., Hansen, H.: Stubborn Set Intuition Explained. ToPNoC **10470**(12), 140–165 (2017). https://doi.org/10.1007/978-3-662-55862-1