International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-022-00672-0

GENERAL

Special Issue: TACAS 2020

Partial-order reduction for parity games and parameterised Boolean equation systems

Thomas Neele[1] · **Tim A. C. Willemse**[1] · **Wieger Wesselink**[1] · **Antti Valmari**[2]

Accepted: 29 August 2022

Abstract
In model checking, reduction techniques can be helpful tools to fight the state-space explosion problem. Partial-order reduction (POR) is a well-known example, and many POR variants have been developed over the years. However, none of these can be used in the context of model checking stutter-sensitive temporal properties. We propose POR techniques for parity games, a well-established formalism for solving a variety of decision problems, including model checking. As a result, we obtain the first POR method that is sound for the full modal μ-calculus. We show how our technique can be applied to the fixed point logic called *parameterised Boolean equation systems*, which provides a high-level representation of parity games. Experiments with our implementation indicate that substantial reductions can be achieved.

Keywords Partial-order reduction · Parity games · Parameterised Boolean equation systems · Stubborn sets

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 blow-up, which is known as the *state-space explosion* problem. Several approaches have been identified to alleviate this issue, by reducing the state space while it is generated. Two established techniques are *symmetry reduction* [17] and *partial-order reduction* (POR) [11,32,37]. Whereas symmetry reduction can only be applied to systems that contain several copies of a component, POR also applies to heterogeneous systems. The idea behind POR is that, out of many interleavings arising from commutative behaviour of concurrent processes, exploring just one interleaving is often sufficient to draw conclusions on the properties of the system. This is achieved by exploring only a specific subset of the outgoing transitions in every state that is visited.

✉ Thomas Neele
t.s.neele@tue.nl

1 Eindhoven University of Technology, Eindhoven, The Netherlands

2 University of Jyväskylä, Jyväskylä, Finland

However, a major drawback of POR is that most variants at best preserve only stutter-insensitive temporal properties. The application of POR is thus limited to fragments of popular logics, namely LTL or CTL* without the next operator (LTL_{-X}/CTL^*_{-X}) [10,39] or the weak modal μ-calculus [35]. Furthermore, most of the variants of POR that preserve the semantics of formulae in a branching time logic impose significant restrictions on the reduction. This decreases the amount of reduction achieved.

In previous work [30], we addressed these shortcomings by applying POR on parity games. A parity game is an infinite-duration, two-player game played on a directed graph with decorations on the nodes, in which the players *even* (denoted ◇) and *odd* (denoted □) strive to win the nodes of the graph. Parity games find common application in model checking, where every node v in the parity game encodes whether a state s in a transition system satisfies a subformula φ of a given formal property, formulated in a temporal logic such as the modal μ-calculus [20]. Under a typical encoding, player ◇ wins in the node 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. We investigated these ideas in [30] and proposed a number of conditions with the aim of preserving the winning player after reduction. Furthermore, we identi-

Published online: 02 October 2022

fied a typical structure that occurs in the setting of model checking and exploited this to improve the reduction potential. We showed how this POR technique can be applied in the context of solving a *parameterised Boolean equation system* (PBES) [13]—a fixed point logic closely related to LFP—as a high-level representation of a parity game. Finally, we extended the implementation ideas of [22] with support for non-determinism and applied these to create an experimental implementation, which we used to evaluate our ideas.

After [30] was presented, Antti Valmari noticed an issue with the POR conditions that affects their correctness, that is, the winning player is not necessarily preserved. This problem was subsequently resolved in Thomas Neele's thesis [26] with the introduction of an additional condition called **P**, for **P**layer. The condition **P** prevents a situation where our reduction forces one player to hand control of the game to the other player, effectively giving the latter player a chance to win. The current work extends [30] as follows:

- The POR conditions are amended with the additional condition **P**. We show the correctness with completely reworked proofs (Theorem 1).
- Extended discussion of the intricacies of applying POR to PBESs. In traditional POR approaches, transition labels are used to determine which transitions should be grouped together. However, this information is lost in the construction of a PBES; grouping related transitions thus relies on heuristics. We show how the choice for such a heuristic may impact reduction.
- We include full proofs for the lemmata that show that our implementation satisfies the POR conditions we set.

Our approach has two distinct benefits over traditional POR techniques that operate on transition systems. First, it is the first work that enables the use of partial-order reduction for model checking for the full modal μ-calculus. Second, the conditions that we propose are strictly weaker than those necessary to preserve the branching structure of a transition system used in other approaches to POR for branching time logics [10,35], increasing the effectiveness of POR.

In our implementation of POR for PBESs, it is necessary to first conduct static analysis on the PBES to identify reduction opportunities. Both this preparatory step and the additional computation required during the exploration phase create overhead compared to a regular exploration procedure. Our experiments show, however, that particularly those instances in which PBESs encode model checking problems involving large state spaces benefit from the use of partial-order reduction. In such cases, a significant size reduction is possible, even when checking complex μ-calculus formulae, and the time overhead mentioned above is more than made up for by the reduction in the number of states.

Related Work The literature is rich with many variants of partial-order reduction; the most prominent are *ample sets* [32], *persistent sets* [11] and *stubborn sets* [37]. These methods are conceptually very similar, although the stubborn set method is the only one that allows reasoning about disabled actions (which label the transition relation), and thus offers more reduction potential. We discuss several extensions and applications of these methods.

There are several proposals for the application of POR in the context of verifying branching-time logics. Groote and Sellink [12] define several forms of *confluence reduction* and prove which behavioural equivalences (and by extension, which fragments of logics) are preserved. In confluence reduction, one tries to identify internal transitions, typically labelled with the action τ that can safely be prioritised, leading to a smaller state space. Ramakrishna and Smolka [35] propose a notion that coincides with strong confluence from [12], preserving weak bisimilarity and the corresponding logic, the weak modal μ-calculus.

Similar ideas are presented by Gerth et al. [10]. Their approach is based on the *ample sets* method [32] and preserves a relation that they call visible bisimulation and the associated logic CTL$_{-X}$. To preserve the branching structure, they introduce a *singleton proviso* which, contrary to our theory, can greatly impair the amount of reduction that can be achieved (see our Example 3, p. 6).

An approach that does preserve a branching semantics but does not need a singleton proviso is proposed by Valmari and Vogler [41]. Their POR conditions are sufficient for preservation of *fair testing equivalence*, which is the weakest congruence that preserves livelocks that cannot be exited. This method is thus suited for checking properties under a fairness assumption.

Valmari [38] (see [40] for an up-to-date discussion) describes the *stubborn sets* method for LTL$_{-X}$ model checking. While investigating the use of stubborn sets for parity games, we identified a subtle issue in one of the stubborn set conditions (called **D1** in [40]). When applied to labelled transition systems or Kripke structures, this means that LTL$_{-X}$ is not necessarily preserved. Moreover, using the condition in the setting of parity games may result in games with different winners; see Example 2. In [27], we further explore the consequences of the faulty condition for stubborn-set-based POR techniques that can be found in the literature. We here resort to a strengthened version of condition **D1** that does not suffer from these issues.

Peled [33] applies POR on the product of a transition system and a Büchi automaton, which represents an LTL$_{-X}$ property. The resulting product automaton thus encodes both the transition system and the property, in a way similar to parity games, resulting in a POR approach that is similar to ours. It is important to note, though, that this original theory is not sound, as discussed in [36]. Kan et al. [18] improve on

Peled's ideas and manage to preserve all of LTL. To achieve this, they analyse the Büchi automaton that corresponds to the LTL formula to identify which part is stutter insensitive. With this information, they can reduce the state space in the appropriate places and preserve the validity of the LTL formula under consideration.

The work of Bønneland et al. [3,4] is close to ours in spirit: they apply POR to reachability games. Such games can be used for synthesis and for model checking reachability properties. The formalism they consider thus has less expressive power than the parity games we consider. Their technique is applied to weighted Petri net games with inhibitor arcs, which are more low-level than our PBESs, and hence some intricacies related to PBESs (see Sect. 4) are avoided. As we already discussed in [30], the conditions proposed in [3] contain a small correctness issue that was later resolved in [4].

Outline We give a cursory overview of parity games in Sect. 2. In Sect. 3 we introduce partial-order reduction for parity games and prove its correctness. A further improvement is introduced in Sect. 3.3. Section 4 briefly introduces the PBES fixed point logic, and in Sect. 5, we describe how to effectively implement parity-game based POR for PBESs. We present the results of our experiments of using parity-game based POR for PBESs in Sect. 6. We conclude in Sect. 7.

2 Preliminaries

Parity games are infinite-duration, two-player games played on a directed graph. The objective of the players, called *even* (denoted by $\diamond$) and *odd* (denoted by $\Box$), is to win nodes in the graph.

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

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

We write $s \to t$ whenever $(s, t) \in E$. The set of successors of a node s is denoted with $succ(s) = \{t \mid s \to t\}$. We use $\bigcirc$ to denote an arbitrary player and $\overline{\bigcirc}$ to denote its opponent. Furthermore, we write $V_\bigcirc = \{v \mid \mathcal{P}(v) = \bigcirc\}$ for the set of nodes that belong to $\bigcirc$.

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 an infinite 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 $\Box$[1]. Note that at least one priority occurs infinitely often on π because Ω is bounded. We stress that we consider *min parity games* (smaller priorities dominate larger priorities); *max parity games* (larger priorities dominate) are also commonly found in literature.

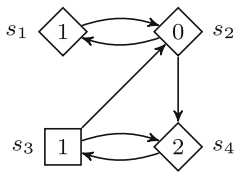


Fig. 1 Example of a parity game

To reason about moves that a player may want to take, we use the concept of *strategies*. A strategy $\sigma_\bigcirc : V^* \cdot V_\bigcirc \to V$ for player $\bigcirc$ is a function that determines where $\bigcirc$ moves the token next, after the token has passed through a finite sequence of nodes. More formally, for all non-empty paths $s_1 \ldots s_n$ such that $\mathcal{P}(s_n) = \bigcirc$, it holds that $\sigma_\bigcirc(s_1 \ldots s_n) \in succ(s_n)$. A play $s_1, s_2, \ldots$ is *consistent* with a strategy σ if and only if $\sigma(s_1 \ldots s_i) = s_{i+1}$ for all i such that $\mathcal{P}(s_i) = \bigcirc$. 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 1 Consider the parity game in Fig. 1. Here, priorities are inscribed in the nodes and the nodes are shaped according to their owner ($\diamond$ or $\Box$). Let π be an arbitrary, possibly empty, sequence of nodes. In this game, the strategy $\sigma_\diamond$, partially defined as $\sigma_\diamond(\pi s_1) = s_2$ and $\sigma_\diamond(\pi s_2) = s_1$, is winning for $\diamond$ in s_1 and s_2. After all, the minimal priority that occurs infinitely often along $(s_1 s_2)^\omega$ is 0, which is even. Player $\Box$ can win node s_3 with the strategy $\sigma_\Box(\pi s_3) = s_4$. Note that player $\diamond$ is always forced to move the token from node s_4 to s_3. □

The strategies $\sigma_\diamond$ and $\sigma_\Box$ used in the above example are *memoryless*; a strategy σ is called memoryless if and only if $\sigma(\pi s)$ is equivalent for all histories π. In fact, memoryless strategies are sufficient for determining the winner of a node: as shown by Zielonka [42], for all players $\bigcirc$, parity games G and nodes s, there is a winning $\bigcirc$-strategy for s in G if and only if there is a winning *memoryless* $\bigcirc$-strategy for s in G. In the remainder, we mostly use memoryless strategies and specify them as a partial function $\sigma : V \to V$. Only in the proof of Theorem 1, we apply strategies with memory.

[1] For this reason, the players are sometimes also referred to as player *even* ($\diamond$) and player *odd* ($\Box$) in the literature.

3 Partial-order reduction

In model checking, arbitrary interleaving of concurrent processes can lead to a combinatorial explosion of the state space. By extension, parity games that encode model checking problems for concurrent processes suffer from the same phenomenon. *Partial-order reduction* (POR) techniques help combat the blow-up. The current work is based on Valmari's theory of stubborn sets [37] and its extension to LTL_{-X} model checking [38] as it can easily deal with nondeterminism [39].

The name "partial-order reduction" came from the idea that if two events occur concurrently, then it is artificial to say that one of them occurred before the other. Therefore, the "occurred before" relation is thought of as a partial order. Permutations of concurrent events span equivalence classes of executions. According to this idea, to analyse the behaviour of the system, it suffices to represent each equivalence class by a single execution. This intuition has proven misleading. When constructing representatives that are kept for executions that are not kept, most "partial order" methods not only permute, but also remove and add events. Furthermore, the property being verified has a strong influence on what needs to be kept. However, the term "partial order" has persisted.

3.1 Weak stubborn sets

Partial-order reduction relies on edge labels, here referred to as *events* and typically denoted with the letter a. In a typical application of POR, such events and edge labelling are deduced from a high-level syntactic description of the graph structure (see also Sect. 4). A *reduction function* subsequently uses these events when producing an equivalent *reduced* graph structure from the same high-level description, such that the answer to the verification question is not changed. For now, we tacitly assume the existence of a set of events and edge labelling for parity games and refer to the resulting structures as *labelled parity games*. For the purpose of defining labelled parity games as a structure on top of parity games, we define edge labels in a somewhat unusual way. This does not affect the theory in any way.

Definition 2 A *labelled parity game* is a triple $L = (G, \mathcal{A}, \ell)$, where $G = (V, E, \Omega, \mathcal{P})$ is a parity game, $\mathcal{A}$ is a set of events and $\ell : \mathcal{A} \to 2^E$ is an edge labelling; it is required that for all $(s,t) \in E$ there is at least one $a \in \mathcal{A}$ such that $(s,t) \in \ell(a)$.

For the remainder of this section, we fix an arbitrary labelled parity game $L = (G, \mathcal{A}, \ell)$. We write $s \xrightarrow{a} t$ whenever $s \to t$ and $(s,t) \in \ell(a)$. The same notation extends to labelled paths, which we call *executions*: $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$. We omit the intermediate states when they are clear from the context or not relevant and write $s \xrightarrow{a_1 a_2 \ldots}$ (or $s \xrightarrow{a_1 \ldots a_2} t$ for finite executions). We say an event a is enabled in a node s, notation $s \xrightarrow{a}$, if and only if there is a transition $s \xrightarrow{a} t$ for some t. The set of all enabled events in a node s is denoted with $\mathit{enabled}_L(s)$.

The set of events $\mathcal{A}$ has a subset $\mathcal{I}$ whose elements are called *invisible*, such that for every $a \in \mathcal{I}$, $s \xrightarrow{a} t$ implies $\mathcal{P}(s) = \mathcal{P}(t)$ and $\Omega(s) = \Omega(t)$ for all $s, t \in V$. The elements of $\mathcal{A} \backslash \mathcal{I}$ are *visible*. This definition allows an implementation to over-approximate $\mathcal{A} \backslash \mathcal{I}$, *i.e.*, if it is too difficult to find out whether $s \xrightarrow{a} t$ implies $\mathcal{P}(s) = \mathcal{P}(t)$ and $\Omega(s) = \Omega(t)$ for all $s, t \in V$, then a may be classified as visible.

A *reduction function* indicates which edges are to be explored in each node, based on the events associated to the edges. Given some initial node $\hat{s}$, such a function induces a unique *reduced labelled parity game* as follows.

Definition 3 Given a node $\hat{s} \in V$ and a *reduction function* $r : V \to 2^{\mathcal{A}}$, the *reduced labelled parity game* induced by r and starting from $\hat{s}$ is defined as $L_r = (G_r, \mathcal{A}, \ell_r)$, where $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$ and ℓ_r are such that:

- if we let $E' = \{(s,t) \in E \mid \exists a \in r(s).(s,t) \in \ell(a)\}$ be the transition relation under r, then $V_r = \{s \mid \hat{s} E'^* s\}$ is the set of nodes reachable with E', where E'^* is the reflexive transitive closure of E', and $E_r = E' \cap (V_r \times V_r)$ is the restricted transition relation;
- Ω_r and $\mathcal{P}_r$ are the respective restrictions of Ω and $\mathcal{P}$ on V_r;
- $\ell_r(a) = \ell(a) \cap E_r$ for all $a \in \mathcal{A}$.

Note that a reduced labelled parity game is only well-defined when $r(s) \cap \mathit{enabled}_L(s) \neq \emptyset$ for every node $s \in V_r$; if this property does not hold, E_r is not total. Even if totality of E_r is guaranteed, the same node s may be won by different players in L and L_r if no restrictions are imposed on r. In case the reduced game is finite, the following conditions on r, as we will show, are sufficient to ensure both. Below, we say an event a is a *key event* in s iff for all executions $s \xrightarrow{a_1 \ldots a_n} s'$ such that $a_1 \notin r(s), \ldots, a_n \notin r(s)$, we have $s' \xrightarrow{a}$. (The underlying intuition will be explained later.) Key events are typically denoted a_{key}. Note that every key event is enabled, by taking $n = 0$.

Definition 4 Given a labelled parity game $L = (G, \mathcal{A}, \ell)$, where $G = (V, E, \Omega, \mathcal{P})$, we say that a reduction function

$r : V \to 2^{\mathcal{A}}$ is a *weak stubborn set* for L iff for all nodes $s \in V$, the following conditions hold[2]:

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

D2w $r(s)$ contains a key event in s.

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 a, every cycle in the reduced game contains a node s' such that $a \in r(s')$.

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

Below, we also use the term (weak) stubborn set to refer to the set of events $r(s)$ in some node s. Originally [37], the set $r(s)$ was constructed so that, if there is an enabled event, it always contained an enabled event that could not be disabled by events outside $r(s)$. This is the origin of the word "stubborn" in its name: at least one event in $r(s)$ is determined to happen and cannot be prevented by the outside world.

We sketch the intuition behind the stubborn set conditions. Condition **D1** ensures that whenever an enabled event is selected for the stubborn set, it does not disable executions not in $r(s)$. Hence, executions that are removed from the state space can still be mimicked, albeit with a slightly different order of events. (The events a and $a_1, \ldots, a_n$ are commuted.) The key event required by **D2w** must be enabled (as mentioned above), which guarantees totality of E_r. Furthermore, condition **D1** applies in particular to key events, and thus executions that are removed can always be mimicked via a key event. In this sense, **D1** and **D2w** together capture the core idea of POR: reduce the number of similar interleavings that are explored. In a traditional setting where POR is applied on a transition system, the combination of **D1** and **D2w** is sufficient to preserve deadlocks, hence their name.

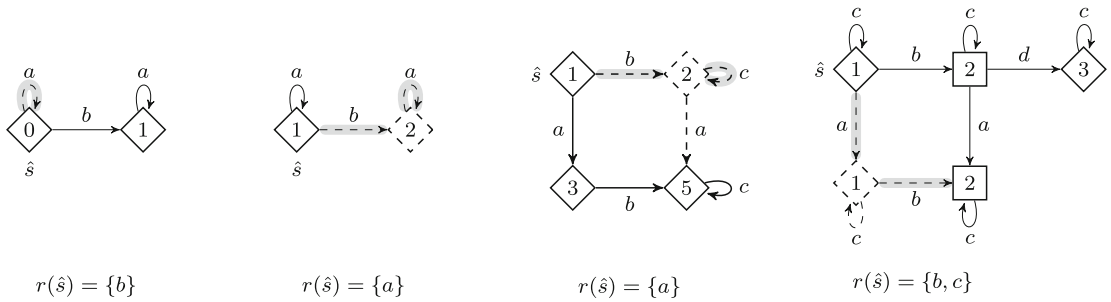
To preserve the winning player in a parity game, we, furthermore, require conditions **V**, **I** and **L**, which originate from the LTL$_{-X}$-preserving stubborn set method [38], and condition **P**, which is specific for parity games. Condition **V** ensures that reduction only occurs when the mimicking execution produced by **D1** does not reorder visible events compared to the executions that are pruned from the state space. Condition **L** prevents the so called *action-ignoring problem*, where a certain visible event is never selected for the stubborn set and ignored indefinitely. Since we assumed the reduced game is finite, such indefinite ignoring can only happen on cycles; this is thus properly addressed by **L**. Condition **I** ensures that if the game has an invisible infinite execution from some state, then also the reduced game has one.

In the traditional setting of POR for transition systems, the LTL$_{-X}$ method using conditions **D1**, **D2w**, **V**, **I** and **L** was designed to show that each infinite (or deadlocking) execution has a stuttering-equivalent representative in the reduce transition system. Our parity game setting adds to this the problem that control of the token may move from one player to another. We thus need the additional condition **P**; this guarantees that the winning player is preserved in roughly the following way. Let s be a node that is won by $\bigcirc$ in the original game and π an execution that is consistent with a winning strategy for $\bigcirc$ in s. When the loser $\overline{\bigcirc}$ is in control of the token in the reduced game, wherever it moves the token next, the token will stay in nodes that $\overline{\bigcirc}$ loses in the original game. On the other hand, when the winner $\bigcirc$ is in control of the token, the LTL$_{-X}$ arguments for representative paths apply until $\bigcirc$ loses control. When that happens, condition **P** ensures that all original events chosen by $\bigcirc$ in π have been executed also in the reduced game. Invisible events may have been added compared to the original execution π, but the $\overline{\bigcirc}$-node t that the token ends up in can also be reached by $\overline{\bigcirc}$ in the original game by following those same invisible events. Hence, the winner $\bigcirc$ also wins t in the original game. Thus, as sketched above, we are able to reduce those parts the game that belong to a single player, and where multiple paths exist that observe the same priorities (up to stuttering).

We use the example below to further illustrate the purpose of—and need for—conditions **V**, **I**, **L** and **P**. In particular, the example illustrates that the winning player in the original game and the reduced game might be different if one of these conditions is not satisfied.

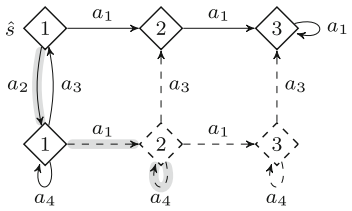
Example 2 First, see the labelled parity games of Fig. 2. These four games show a reduced game under a reduction function satisfying **D1** and **D2w** but not **I**, **L**, **V** or **P**, respectively. In each case, we start exploration from the node called $\hat{s}$, using the indicated reduction function in $\hat{s}$; for all other nodes s' we have $r(s') = \mathcal{A}$. This yields the respective games that only contain the solid nodes and edges; the dashed parts have been pruned. Consequently, the winning strategy for player $\diamond$ in the original game (highlighted in grey in the figures) is lost. The example in Fig. 2d, showing the necessity of condition **P**, is due to Antti Valmari. Our previous work [30] did not contain condition **P** and may thus fail to preserve the winner in this game.

[2] As noted before, the condition **D1** that we propose is stronger than the version in the literature [38,40] since that one suffers from the *inconsistent labelling problem* [27,28] which also manifests itself in the parity game setting, as we will demonstrate in Example 2.



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

Fig. 2 Four games that show the winner is not necessarily preserved if we drop one of the conditions **V**, **I**, **L** or **P**. 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 highlighted in grey



D1, D2w, V, I, S, P

Fig. 3 A parity game that shows condition **L** cannot be weakened to reason about strongly connected components (condition **S**) instead of cycles

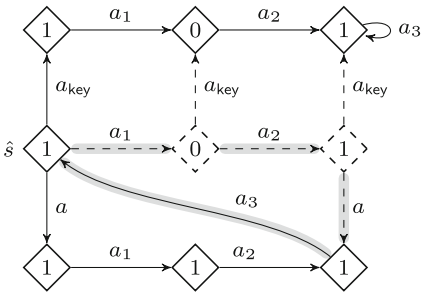


Fig. 4 A parity game that illustrates the inconsistent labelling problem also occurs in parity games when using the original formulation of condition **D1**

Now consider the parity game of Fig. 3, which is inspired by an example from Valmari and Hansen [40]. Here, the condition **L** is replaced by a weaker condition which states that for all visible events a, every *strongly connected component* (SCC) in the reduced game must contain a node s such that $a \in r(s)$. This condition is often called **S** in the literature. Remark that the two leftmost nodes form an SCC and we have $a_1 \in r(\hat{s})$, thus satisfying **S**. However, condition **L** is not satisfied in the cycle consisting of only the bottom-left node.

Finally, consider the parity game in Fig. 4. This game shows that, if instead of condition **D1** as presented in the current work, we use the original formulation [38]

> For all $a \in r(s)$ and $a_1 \notin r(s), \ldots, a_n \notin r(s)$, if $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$, then there are nodes $s', s'_1, \ldots, s'_{n-1}$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s'_n$.

then the winning player is not necessarily preserved. We refer the reader to [28] for an in-depth discussion of this problem in the setting of LTL$_{-X}$ model checking. □

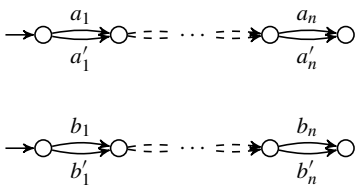
Note that the games in Figs. 2a–c and 3 are from a subclass of parity games called *weak solitaire*, illustrating the need for the identified conditions even in restricted settings. A game is *solitaire* iff at most one player can make non-trivial choices, *i.e.*, the player owns a node with more than one outgoing edge. A game is *weak* iff the priorities along all its paths are non-decreasing, *i.e.*, if $s \to t$ then $\Omega(s) \leq \Omega(t)$. The game in Fig. 2d is weak, but not solitaire. Conversely, the game in Fig. 4 is solitaire, but not weak. Weak solitaire games can encode the model checking of safety properties, solitaire games can capture logics such as LTL and ∀CTL* [14] (the universal fragment of CTL*) and weak games can be used to check CTL properties.

Before we argue for the correctness of our POR approach in the next section, we shortly demonstrate how our approach improves over existing methods for branching time logics. On top of the basic conditions **C1**-**C3**, which preserve LTL$_{-X}$ and are similar in spirit to our conditions **D1** through **L**, Gerth et al. [10] propose the following *singleton proviso*:

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

This extra condition helps preserve the branching structure and is thus needed for preservation of CTL$_{-X}$. However, it can severely impact the amount of reduction achieved; see the following example.

Example 3 Consider the two transition systems below, where $n \geq 1$ is some large natural number. □



The cross-product of these transition systems contains $(n+1)^2$ states.

Assume we want to check the CTL-property $\varphi = \forall\Box\exists\Diamond\, a_n$, where a_n is true in a state if and only action a_n is enabled, on this cross-product. This property expresses: "for all reachable states, there is a path to a state where a_n is enabled", which does not hold. There is no equivalent formula in LTL, and hence, condition **C4** is required for POR to preserve this property.

In the initial state $\hat{s}$, neither $r(\hat{s}) = \{a_1, a'_1\}$ nor $r(\hat{s}) = \{b_1, b'_1\}$ is a valid stubborn set, due to **C4**. Other choices for $r(\hat{s})$ are ruled out by the remaining conditions, so we are forced to set $r(\hat{s}) = \mathcal{A}$. A similar argument applies to all other states and thus no reduction can be achieved.

Alternatively, we can choose to attack our model checking problem by encoding it in a parity game. For this, we use a μ-calculus formula which corresponds to φ, namely $\nu X.([-]X \wedge \mu Y.(\langle - \rangle Y \vee \langle a_n \rangle \mathit{true}))$. The resulting labelled parity game is depicted in Fig. 5. When not explicitly indicated, horizontal edges are labelled with events b_i and b'_i (and therefore drawn twice), vertical edges are labelled with events a_i and a'_i and diagonal edges are labelled with c. This game contains $2(n+1)^2 + 1$ nodes: there is one node for each combination of state (of which there are $(n+1)^2$) and fixpoint variable (X and Y) in the formula, and one auxiliary node.

Applying POR with our conditions **D1** through **P** allows choosing $r(\hat{v}) = \{b_1, b'_1\}$ in the initial node $\hat{v}$; an analogous choice can be made in the other nodes. This ultimately results in a game of $3(n+1)$ nodes, which we can solve to answer our model checking question.

While several optimisations for CTL$_{-X}$ model checking under POR are proposed in [24], unlike our approach, those optimisations can only be applied to certain classes of CTL$_{-X}$ formulas and not in general.

3.2 Correctness

Condition **D2w** suffices, as we already argued, to ensure totality of the transition relation of the reduced labelled parity game. Hence, we are left to argue that the reduced game preserves and reflects the winner of the preserved nodes of the original game; this is formally claimed in Theorem 1. We do so by constructing a strategy in the reduced game

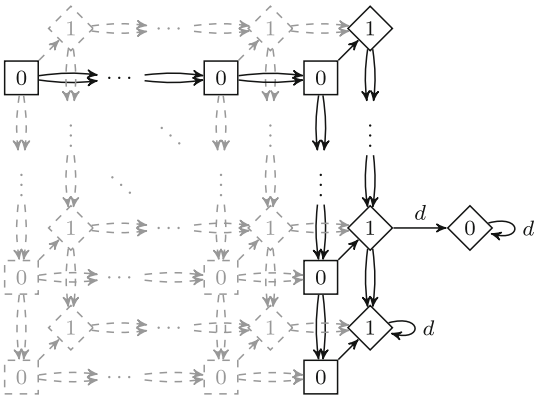


Fig. 5 Parity game corresponding to the CTL model checking problem of Example 3

that mimics the winning strategy in the original game. We use *stutter equivalence* (formally defined below) to compare plays that follow these strategies and use this to conclude that the mimicking strategy we constructed is winning in the reduced game. This reasoning is based on [9].

We introduce a couple of auxiliary lemmata, required for our main correctness theorem. Fix a labelled parity game $L = (G, \mathcal{A}, \ell)$ with $G = (V, E, \Omega, \mathcal{P})$, a node $\hat{s}$, a weak stubborn set r and the reduced labelled parity game $L_r = (G_r, \mathcal{A}, \ell_r)$, with $G_r = (V_r, E_r, \Omega_r, \mathcal{P}_r)$, induced by r and $\hat{s}$. We assume r and $\hat{s}$ are such that V_r is of finite size. Given a path $\pi = s_0 s_1 s_2 \ldots$ (either in G or G_r), we define the *no-stutter trace*, notation $\mathsf{no\text{-}stut}(\pi)$, as the sequence of those $\Omega(s_i)$ such that $i = 0$ or $\Omega(s_i) \neq \Omega(s_{i-1})$.

Definition 5 Let $\pi = s_0 s_1 s_2 \ldots$ and $\rho = t_0 t_1 t_2 \ldots$ be two paths in G. We say π and ρ are stutter equivalent, notation $\pi \triangleq \rho$, if and only if they are both finite or both infinite, and $\mathsf{no\text{-}stut}(\pi) = \mathsf{no\text{-}stut}(\rho)$.

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

Proof Let $\pi = s_0 s_1 s_2 \ldots$ be an infinite path and p the minimal priority that occurs infinitely often on π. Take an arbitrary infinite path ρ such that $\mathsf{no\text{-}stut}(\pi) = \mathsf{no\text{-}stut}(\rho)$.

If $\mathsf{no\text{-}stut}(\pi)$ is finite, then p must be its last element and p is in fact the only priority that occurs infinitely often on π. The same then applies to ρ: p is the only priority that occurs infinitely often on ρ.

Otherwise, if $\mathsf{no\text{-}stut}(\pi)$ is infinite, then it is equal to $\Omega(s_0)\Omega(s_1)\Omega(s_2)\ldots$ where finite repetitions are collapsed. This operation does not influence which priorities occur infinitely often. Hence, p is also the minimal priority that occurs infinitely often in $\mathsf{no\text{-}stut}(\rho)$ and, consequently, on ρ.

Since p is the minimal priority that occurs infinitely often on both π and ρ, they must have the same winner. □

Our second lemma provides a basic stubborn set fact on the availability of key transitions.

Lemma 2 *If, for some event a, it holds that $s \xrightarrow{a}$ and $a \notin r(s)$, then $r(s)$ contains an invisible key event in s.*

Proof If a is invisible, $r(s)$ contains at least one invisible key event, due to **I**.

Otherwise, if a is visible, we obtain a key event $a_{\mathsf{key}} \in r(s)$ from **D2w**. In case a_{key} is visible, we violate condition **V** with the assumption that $a \notin r(s)$ and since a_{key} is enabled by the key event property. We conclude that a_{key} is invisible. □

In the lemmata below, we write $\rightarrow_r$ to stress which transition must occur in G_r.

Lemma 3 [28, Lemma 5.1] *If **D1** yields $\rho = s \xrightarrow{a a_1 \dots a_n} s'$ from $\pi = s \xrightarrow{a_1 \dots a_n a} s'$, then $\pi \triangleq \rho$.*

Proof Assume the executions are of the shape $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n \xrightarrow{a} s'_n$ and $\rho = s_0 \xrightarrow{a} s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s'_n$.

If a is invisible, then it follows from **D1** that $s_i \xrightarrow{a} s'_i$ for all $1 \leq i < n$. With the transitions $s_0 \xrightarrow{a} s'_0$ and $s_n \xrightarrow{a} s'_n$ that we already had, we obtain $\Omega(s_i) = \Omega(s'_i)$ for all $0 \leq i \leq n$. It follows that $\pi \triangleq \rho$.

If a is visible, then our assumption that $a \in r(s_0)$ means that $r(s_0)$ contains an enabled visible event, and, by **V**, it contains all visible events. Thus the events $a_1, \dots, a_n$, which we assumed are not in $r(s_0)$, must be invisible. We get $\Omega(s_0) = \Omega(s_1) = \dots = \Omega(s_n)$ and $\Omega(s'_0) = \Omega(s'_1) = \dots = \Omega(s'_n)$. If $\Omega(s_0) \neq \Omega(s'_0)$, we reason that

$$\mathsf{no\text{-}stut}(\pi) = \Omega(s_0)\Omega(s'_n) = \Omega(s_0)\Omega(s'_0) = \mathsf{no\text{-}stut}(\rho).$$

Otherwise, $\mathsf{no\text{-}stut}(\pi) = \Omega(s_0) = \mathsf{no\text{-}stut}(\rho)$ holds trivially. We conclude $\pi \triangleq \rho$. □

Lemma 4 *Let $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots$ be any execution such that $a_i \notin r(s_0)$ for every a_i occurring on π. Then there is an execution $\rho = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots$ for some invisible event a_{key}, and it holds that $\pi \triangleq \rho$.*

Proof Let $a_{\mathsf{key}} \in r(s_0)$ be an invisible key event, obtained through Lemma 2 by $s_0 \xrightarrow{a_1}$ and $a_1 \notin r(s_0)$. The key event property gives us, for each i, a state $s'_{i,i}$ such that $s_i \xrightarrow{a_{\mathsf{key}}} s'_{i,i}$.

In case π has finite length n, we can simply apply **D1** to the execution $s_0 \xrightarrow{a_1 \dots a_n} s_n \xrightarrow{a_{\mathsf{key}}} s'_{n,n}$ to obtain $\rho = s_0 \xrightarrow{a_{\mathsf{key}} a_1 \dots a_n} s'_{n,n}$. By Lemma 3, we have $\pi \triangleq \rho$.

If π is infinite, we use a reasoning similar to that of König's Lemma [21]. We can apply **D1** to obtain a finite execution $\pi_i = s_0 \xrightarrow{a_{\mathsf{key}}} s'_{i,0} \xrightarrow{a_1} \dots \xrightarrow{a_i} s'_{i,i}$ for every $i \geq 0$. Because

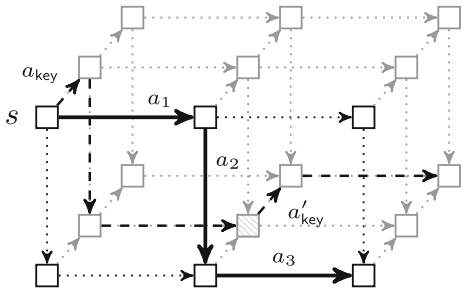


Fig. 6 Example of how a_1, a_2, a_3 is mimicked by introducing a_{key} and a'_{key} and moving a_2 to the front (dashed trace). Transitions that are drawn in parallel have the same label

a_{key} is invisible, **D1** furthermore gives us $s_k \xrightarrow{a_{\mathsf{key}}} s'_{i,k}$ for every $i \geq 0$ and $1 \leq k < i$. We prove by induction that for every k, there is s'_k such that $s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0$ (for $k = 0$) or $s'_{k-1} \xrightarrow{a_k} s'_k$ (for $k > 0$), and $s'_k = s'_{i,k}$ for infinitely many values of i.

Because there are only finitely many states, there is a state s'_0 that is the same as $s'_{i,0}$ for infinitely many values of i. This constitutes the base case.

To prove the induction step, we observe that all or all but one of the infinitely many i with $s'_{i,k} = s'_k$ satisfy $i > k$, and thus have an $s'_{i,k+1}$ such that $s'_k \xrightarrow{a_{k+1}} s'_{i,k+1}$. Infinitely many of these $s'_{i,k+1}$ are the same state, again because there are only finitely many states. This state qualifies as s'_{k+1}.

Then we define $\rho = s_0 \xrightarrow{a_{\mathsf{key}}}_r s'_0 \xrightarrow{a_1} s'_1 \xrightarrow{a_2} \dots$. Since a_{key} is invisible, we have $\Omega(s_j) = \Omega(s'_j)$ for every $j \geq 0$. This implies $\pi \triangleq \rho$. □

We remark that Lemma 4 also holds for reduced labelled parity games that have an infinite state space, but where all the events are finitely branching. Before we continue with the next lemmata, we first consider a short example to provide some intuition on how Lemmata 3 and 4 can be applied to construct alternative paths in the reduced game.

Example 4 The structure of Fig. 6, in which parallel edges have the same label, visualises part of a game in which the execution $\pi = s \xrightarrow{a_1 a_2 a_3}$ (drawn with solid edges) is part of a play that traverses nodes of player □. If we assume $a_1 \notin r(s)$, then π does not exist in the reduced game. However, it might be possible to mimic π through the path that follows the edges $a_{\mathsf{key}} a_2 a_1 a'_{\mathsf{key}} a_3$ (drawn with dashes). The new play reorders the events a_1, a_2 and a_3 according to the construction of Lemma 3 and introduces the key events a_{key} and a'_{key} according to the construction of Lemma 4. Note that existence of a'_{key} is only guaranteed if $a_3 \notin r(\boxtimes)$ (the stubborn set in the fourth node of the mimicking path). □

We now continue with two lemmata that, respectively, show how finite and infinite executions from the original game can be mimicked in the reduced game.

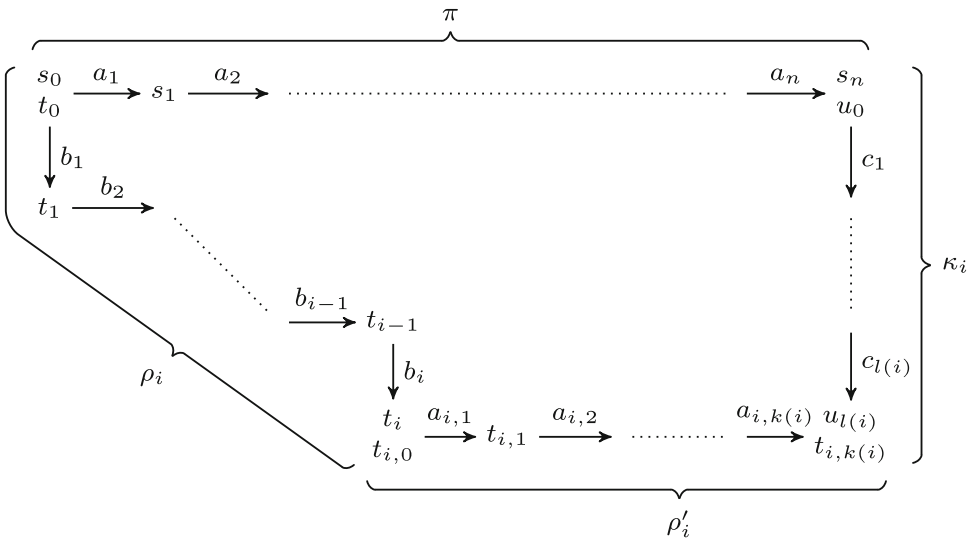


Fig. 7 Showing the relation between the paths π, ρ_i, ρ'_i and π_i from the proof of Lemma 5

Lemma 5 *If $s_0 \in V_r$ and $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$, where $\mathcal{P}(s_0) = \dots = \mathcal{P}(s_{n-1}) \neq \mathcal{P}(s_n)$, then there are executions $\rho := s_0 = t_0 \xrightarrow{b_1}_r t_1 \xrightarrow{b_2}_r \dots \xrightarrow{b_m}_r t_m$, where $\mathcal{P}(t_0) = \dots = \mathcal{P}(t_{m-1}) \neq \mathcal{P}(t_m)$, and $\kappa = s_n \xrightarrow{c_1 \dots c_m} t_m$ such that $\pi \triangleq \pi\kappa \triangleq \rho$.*

Proof We use induction to prove, for some m and each $0 \leq i \leq m$, the existence of the following executions for some $k(i)$, where $l(i) = i + k(i) - n$:

$$\rho_i = t_0 \xrightarrow{b_1}_r t_1 \xrightarrow{b_2}_r \dots \xrightarrow{b_i}_r t_i$$
$$\rho'_i = t_{i,0} \xrightarrow{a_{i,1}} t_{i,1} \xrightarrow{a_{i,2}} \dots \xrightarrow{a_{i,k(i)}} t_{i,k(i)}$$
$$\kappa_i = u_0 \xrightarrow{c_1} u_1 \xrightarrow{c_2} \dots \xrightarrow{c_{l(i)}} u_{l(i)}$$

These executions, and also π, are connected; we have $s_0 = t_0$, $t_i = t_{i,0}$, $u_0 = s_n$ and $u_{l(i)} = t_{i,k(i)}$ (see Fig. 7). Below, let $\mathcal{P}(\rho_i \rho'_i)$ be the sequence of node owners observed along $\rho_i \rho'_i$. The induction assumption, furthermore, contains:

$$\mathsf{no\text{-}stut}(\pi) = \mathsf{no\text{-}stut}(\pi\kappa_i) \tag{1}$$
$$\mathsf{no\text{-}stut}(\rho_i \rho'_i) = \mathsf{no\text{-}stut}(\pi\kappa_i) \tag{2}$$
$$\mathcal{P}(\rho_i \rho'_i) = \mathcal{P}(s_0)^{i+k(i)-1} \mathcal{P}(s_n) \tag{3}$$

The intuition behind the hypothesised executions is as follows: ρ_i contains transitions of π that we have been able to mimic in the reduced game after i steps, as well as key events that were necessarily introduced along the way. Those same key events can also be used to extend π in the original game, this is the execution κ_i. The transitions from π that are still to be mimicked make up ρ'_i.

The base case $i = 0$ of the induction is obtained by setting ρ_0 and κ_0 to the empty execution and $\rho'_0 = \pi$. By $\pi = \pi\kappa_0$, (1) holds. Thanks to $\rho_0\rho'_0 = \pi\kappa_0$, (2) and (3) also hold.

For the induction step, we distinguish two cases when $k(i) > 0$. The situation where $k(i) = 0$ will be handled at the end of the proof.

If at least one of $a_{i,1}, \dots, a_{i,k(i)}$ is in $r(t_i)$, then **D1** can be applied to the first such $a_{i,j}$, yielding $t_i \xrightarrow{a_{i,j}}_r t_{i+1} \xrightarrow{a_{i,1} \dots a_{i,j-1} a_{i,j+1} \dots a_{i,k(i)}} t_{i,k(i)}$. This specifies t_{i+1}, and we choose $b_{i+1} = a_{i,j}$ and $a_{i+1,1} \dots a_{i+1,k(i+1)} = a_{i,1} \dots a_{i,j-1}$ $a_{i,j+1} \dots a_{i,k(i)}$. We call this "moving $a_{i,j}$ to the front". Execution κ_{i+1} remains unchanged from κ_i, since ρ'_{i+1} still ends in $t_{i,k(i)} = t_{i+1,k(i+1)}$ (and also $l(i+1) = l(i)$, since $k(i+1) = k(i) - 1$). Using (1) from the induction hypothesis, it thus follows that $\mathsf{no\text{-}stut}(\pi) \overset{(1)}{=} \mathsf{no\text{-}stut}(\pi\kappa_i) = \mathsf{no\text{-}stut}(\pi\kappa_{i+1})$, and (1) also holds for $i+1$.

For (2), we apply Lemma 3 and part of the induction hypothesis to deduce

$$\begin{aligned}
&\mathsf{no\text{-}stut}(t_0 \xrightarrow{b_1 \dots b_{i+1}}_r t_{i+1} \xrightarrow{a_{i+1,1} \dots a_{i+1,k(i+1)}} t_{i+1,k(i+1)}) \\
&= \mathsf{no\text{-}stut}(t_0 \xrightarrow{b_1 \dots b_i}_r t_i \xrightarrow{a_{i,j}} t_{i+1} \xrightarrow{a_{i,1} \dots a_{i,j-1} a_{i,j+1} \dots a_{i,k(i)}} t_{i,k(i)}) \\
&\overset{(L3)}{=} \mathsf{no\text{-}stut}(t_0 \xrightarrow{b_1 \dots b_i}_r t_i \xrightarrow{a_{i,1} \dots a_{i,k(i)}} t_{i,k(i)}) \\
&\overset{(2)}{=} \mathsf{no\text{-}stut}(\pi\kappa_i) \\
&= \mathsf{no\text{-}stut}(\pi\kappa_{i+1})
\end{aligned}$$

Therefore, $i+1$ also satisfies (2).

Finally, for (3), we reason that the last event will not be moved forward unless it is the only event in ρ'_i, *i.e.*, $j = k(i)$ implies $k(i) = 1$. Assume that $a_{i,1}, \dots, a_{i,k(i)-1} \notin r(t_i)$ and $a_{i,k(i)} \in r(t_i)$. By our initial assumption on the owners of nodes in π, we have

$$\mathcal{P}(t_{i,k(i)-1}) \overset{(3)}{=} \mathcal{P}(s_0) \neq \mathcal{P}(s_n) \overset{(3)}{=} \mathcal{P}(t_{i,k(i)})$$

so $a_{i,k(i)}$ must be visible[3] and, by **D1**, enabled in t_i. Condition **V** thus requires that all visible events are in $r(t_i)$; $a_{i,1}, \ldots, a_{i,k(i)-1}$ are not, so they must all be invisible. We can deduce:

$$\mathcal{P}(t_i) \overset{(3)}{=} \mathcal{P}(s_0) \neq \mathcal{P}(s_n) \overset{(3)}{=} \mathcal{P}(t_{i,k(i)}) = \mathcal{P}(t_{i+1})$$

Thus, the antecedent of condition **P** is fulfilled by the transition $t_i \xrightarrow{a_{i,k(i)}}_r t_{i+1}$, and we have to set $r(t_i) = \mathcal{A}$. This then contradicts our assumption that $a_{i,1}, \ldots, a_{i,k(i)-1} \notin r(t_i)$, unless $k(i) = 1$. So, after moving the event $a_{i,j}$ to the front, we must have $\mathcal{P}(\rho_i \rho_i') = \mathcal{P}(\rho_{i+1} \rho_{i+1}')$ for one of the following reasons:

- $a_{i,j}$ is invisible: in this case the vertical transitions $t_{i,v} \xrightarrow{a_{i,j}} t_{i+1,v}$ required by **D1** guarantee $\mathcal{P}(t_{i,v}) = \mathcal{P}(t_{i+1,v})$ for all $0 \leq v < j$. It immediately follows that $\mathcal{P}(\rho_i \rho_i') = \mathcal{P}(\rho_{i+1} \rho_{i+1}')$.
- $a_{i,j}$ is visible and $j < k(i)$: now the events $a_{i,0}, \ldots, a_{i,j-1}$ must be invisible by **V**, since we assumed that they are not contained in $r(t_i)$. This gives us $\mathcal{P}(t_{i,0}) = \cdots = \mathcal{P}(t_{i,j-1})$ and $\mathcal{P}(t_{i+1,0}) = \cdots = \mathcal{P}(t_{i+1,j-1})$. By $j < k(i)$ and assumption (3), we obtain $\mathcal{P}(t_{i,j-1}) = \mathcal{P}(t_{i,j})$, which together with the previous equalities gives us $\mathcal{P}(\rho_i \rho_i') = \mathcal{P}(\rho_{i+1} \rho_{i+1}')$.
- $a_{i,j}$ is visible and $j = k(i)$: as we argued above, this implies that $k(i) = 1$. Thus, $a_{i,j}$ was already at the front and $\rho_i \rho_i'$ and $\rho_{i+1} \rho_{i+1}'$ actually coincide. The result $\mathcal{P}(\rho_i \rho_i') = \mathcal{P}(\rho_{i+1} \rho_{i+1}')$ follows trivially.

We also have $i + k(i) = (i+1) + k(i+1)$, and we conclude that (3) remains valid:

$$\begin{aligned} \mathcal{P}(\rho_{i+1} \rho_{i+1}') &= \mathcal{P}(\rho_i \rho_i') \overset{(3)}{=} \mathcal{P}(s_0)^{i+k(i)-1} \mathcal{P}(s_n) \\ &= \mathcal{P}(s_0)^{(i+1)+k(i+1)-1} \mathcal{P}(s_n) \end{aligned}$$

In the opposite case none of $a_{i,1}, \ldots, a_{i,k(i)}$ is in $r(t_i)$. In that case, we obtain an invisible key event $a_{\mathsf{key}} \in r(t_i)$ from Lemma 2 by $t_i \xrightarrow{a_{i,1}}$ and $a_{i,j} \notin r(t_i)$. The key event property yields the transition $t_{i,k(i)} \xrightarrow{a_{\mathsf{key}}} t_{i+1,k(i+1)}$, to which we can apply Lemma 3 to obtain $t_i \xrightarrow{a_{\mathsf{key}}}_r t_{i+1} \xrightarrow{a_{i,1} \ldots a_{i,k(i)}} t_{i+1,k(i+1)}$. We choose $b_{i+1} = a_{\mathsf{key}}$ and $a_{i+1,1} \ldots a_{i+1,k(i+1)} = a_{i,1} \ldots a_{i,k(i)}$. We call this "introducing a key event". Note that $l(i+1) = l(i) + 1$ (since $k(i+1) = k(i)$), so we need to extend κ_i, for which we use the transition $t_{i,k(i)} \xrightarrow{a_{\mathsf{key}}} t_{i+1,k(i+1)}$, yielding $u_{l(i+1)} = t_{i+1,k(i+1)}$ and $a_{\mathsf{key}} = c_{l(i+1)}$. The invisibility of a_{key} gives us:

$$\mathsf{no\text{-}stut}(\pi) \overset{(1)}{=} \mathsf{no\text{-}stut}(\pi \kappa_i) = \mathsf{no\text{-}stut}(\pi \kappa_{i+1}),$$

and so (1) also holds for $i+1$.

The invisibility of $a_{\mathsf{key}} = b_{i+1} = c_{l(i+1)}$ also allows us to deduce

$$\begin{aligned} &\mathsf{no\text{-}stut}(t_0 \xrightarrow{b_1 \ldots b_{i+1}} t_{i+1} \xrightarrow{a_{i+1,1} \ldots a_{i+1,k(i+1)}} t_{i+1,k(i+1)}) \\ &= \mathsf{no\text{-}stut}(t_0 \xrightarrow{b_1 \ldots b_i} t_i \xrightarrow{a_{i,1} \ldots a_{i,k(i)}} t_{i,k(i)}) \\ &\overset{(2)}{=} \mathsf{no\text{-}stut}(s_0 \xrightarrow{a_1 \ldots a_n} s_n \xrightarrow{c_1 \ldots c_{l(i)}} u_{l(i)}) \\ &= \mathsf{no\text{-}stut}(s_0 \xrightarrow{a_1 \ldots a_n} s_n \xrightarrow{c_1 \ldots c_{l(i)}} u_{l(i)} \xrightarrow{c_{l(i+1)}} u_{l(i+1)}) \end{aligned}$$

So (2) remains valid in the step from i to $i+1$.

Since a_{key} is invisible, ρ_{i+1} is simply extended with one more node belonging to $\mathcal{P}(s_0)$ and, by **D1**, we have $\mathcal{P}(t_{i,j}) = \mathcal{P}(t_{i+1,j})$ for all $0 < j \leq k(i)$, so it follows that (3) remains valid as well:

$$\mathcal{P}(\rho_{i+1} \rho_{i+1}') = \mathcal{P}(s_0)^{i+1+k(i+1)-1} \mathcal{P}(s_n)$$

As the proof of Lemma 6 will show in more detail, due to condition **L** and finiteness of the reduced game, the introduction of a key event can only happen at most $|V_r|$ consecutive times before a cycle is closed and some $a_{i,j}$ necessarily occurs in $r(t_i)$. Thus, we are guaranteed to eventually reach an iteration m such that $k(m) = 0$. In that case, ρ_m' is empty and all events in π have been mimicked, so we can stop the induction. We obtain the required executions $\rho = \rho_m$ and $\kappa = \kappa_m$. □

Lemma 6 *If $s_0 \in V_r$ and $\pi = s_0 \xrightarrow{a_1 a_2 \ldots}$, then there is an execution $\rho = s_0 \xrightarrow{b_1 b_2 \ldots}_r$ such that $\pi \triangleq \rho$.*

Proof We use induction to prove, for each $i \geq 0$:

- the existence of s_i and, if $i > 0$, b_i such that $s_{i-1} \xrightarrow{b_i}_r s_i$; and
- the existence of an execution $s_{i,0} \xrightarrow{a_{i,1}} s_{i,1} \xrightarrow{a_{i,2}} \ldots$, where $s_i = s_{i,0}$.

We assume the following:

$$\mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1 \ldots b_i}_r s_i \xrightarrow{a_{i,1} a_{i,2} \ldots}) = \mathsf{no\text{-}stut}(\pi) \qquad \text{(IH)}$$

Then, ρ is the infinite execution $s_0 \xrightarrow{b_1 b_2 \ldots}_r$.

The base case $i = 0$ of the induction is obtained by choosing $s_0 \xrightarrow{a_{0,1} a_{0,2} \ldots}$, which is equivalent to π. Thanks to $b_1 \ldots b_0 = \varepsilon$, (IH) holds.

Regarding the induction step, if at least one of $a_{i,1}$, $a_{i,2}$, …is in $r(s_i)$, then **D1** can be applied to the first such $a_{i,j}$, yielding $s_i \xrightarrow{a_{i,j}}_r s_{i+1} \xrightarrow{a_{i,1} \ldots a_{i,j-1} a_{i,j+1} \ldots}$, which moves $a_{i,j}$ to the front. This specifies s_{i+1} and for our events we choose

[3] Here it is important that (in)visibility concerns not only priorities, but also node ownership, otherwise we would not be able to derive visibility of $a_{i,k(i)}$.

$b_{i+1} = a_{i,j}$ and $a_{i+1,1}a_{i+1,2}\ldots = a_{i,1}\ldots a_{i,j-1}a_{i,j+1}\ldots$. We apply Lemma 3 and the induction hypothesis to deduce

$$\begin{aligned}
&\mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_{i+1}}_r s_{i+1} \xrightarrow{a_{i+1,1}a_{i+1,2}\cdots})\\
&= \mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_i}_r s_i \xrightarrow{a_{i,j}} s_{i+1} \xrightarrow{a_{i,1}\ldots a_{i,j-1}a_{i,j+1}\cdots})\\
&\overset{(L3)}{=} \mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_i}_r s_i \xrightarrow{a_{i,1}a_{i,2}\cdots})\\
&\overset{(IH)}{=} \mathsf{no\text{-}stut}(\pi)
\end{aligned}$$

Therefore, $i+1$ also satisfies (IH).

In the opposite case none of $a_{i,1}, a_{i,2}, \ldots$ is in $r(s_i)$. Lemma 4 yields s_{i+1} such that $s_i \xrightarrow{a_{\mathsf{key}}}_r s_{i+1} \xrightarrow{a_{i,1}a_{i,2}\cdots}$. We can thus introduce a key event and choose $b_{i+1} = a_{\mathsf{key}}$ and $a_{i+1,1}a_{i+1,2}\ldots = a_{i,1}a_{i,2}\ldots$. Furthermore, invisibility of a_{key} gives

$$\begin{aligned}
&\mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_{i+1}} s_{i+1} \xrightarrow{a_{i+1,1}a_{i+1,2}\cdots}) =\\
&\quad \mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_i} s_i \xrightarrow{a_{i,1}a_{i,2}\cdots})
\end{aligned}$$

So (IH) remains valid in the step from i to $i+1$.

Since we have shown that (IH) holds for every $i \geq 0$, it follows that $\mathsf{no\text{-}stut}(\rho)$ is a prefix of $\mathsf{no\text{-}stut}(\pi)$. To show that $\mathsf{no\text{-}stut}(\rho)$ and $\mathsf{no\text{-}stut}(\pi)$ are in fact equal, we assume that $\mathsf{no\text{-}stut}(\rho)$ is a proper prefix of $\mathsf{no\text{-}stut}(\pi)$ and try to derive a contradiction.

The fact that $\mathsf{no\text{-}stut}(\rho)$ is a proper prefix means that it must be finite, and hence there is an i such that $\mathsf{no\text{-}stut}(\rho) = \mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_i} s_i)$. By the latter fact and the proper prefix property, there is a v such that $\Omega(s_{i,v-1}) \neq \Omega(s_{i,v})$, and so $a_{i,v}$ is visible. We use the smallest such v.

Observe that if **D1** is applied at s_i to move event $a_{i,j}$ to the front, where $j > v$, or **D2w** is applied, then $a_{i+1,k} = a_{i,k}$ for $1 \leq k \leq v$. If the same also happens at s_{i+1} then $a_{i+2,k} = a_{i,k}$ for $1 \leq k \leq v$, and so on, either forever or until **D1** is applied such that $j \leq v$, whichever comes first. We show next that the latter comes first.

Because S_r is finite, we may let $n = i + |S_r|$. By the pigeonhole principle, $s_i, \ldots, s_n$ cannot all be distinct. So the execution $s_i \xrightarrow{b_{i+1}\ldots b_n} s_n$ contains a cycle. **L** implies that there is $i \leq l < n$ such that $a_{i,v} \in r(s_l)$. This guarantees that there is the smallest h such that $i \leq h < i + |S_r|$ and $\{a_{i,1}, \ldots, a_{i,v}\} \cap r(s_h) \neq \emptyset$. Observe that at any step $i \leq i' < h$, whether **D1** is applied to move $a_{i',j}$ forward, where $j > v$, or **D2w** is applied to introduce a key event, we have $a_{i'+1,v} = a_{i',v}$. Furthermore, for all $i \leq i' < h$, **V** and the fact that $a_{i,v} \notin r(s_{i'})$ imply that $b_{i'}$ must be invisible, so the priorities occurring before and after $a_{i,v}$ are preserved, that is, $\Omega(s_{i',v-1}) = \Omega(s_{i'+1,v-1})$ and $\Omega(s_{i',v}) = \Omega(s_{i'+1,v})$. By **D1**, b_{h+1} is one of $a_{h,1}, \ldots, a_{h,v}$. So either $b_{h+1} = a_{i,v}$ or $a_{i,v} = a_{h+1,v-1}$. In the latter case, $s_{i',v-1} = s_{i'+1,v-1}$ and $s_{i',v} = s_{i'+1,v}$.

Repeating the argument at most v times proves that there is $i \leq h < i + v|S_r|$ such that $b_{h+1} = a_{i,v}$. Furthermore, we have $\Omega(s_h) = \Omega(s_{i,v-1}) \neq \Omega(s_{i,v}) = \Omega(s_{h+1})$, which contradicts $\mathsf{no\text{-}stut}(\rho) = \mathsf{no\text{-}stut}(s_0 \xrightarrow{b_1\ldots b_i} s_i)$. □

To be able to lift our knowledge about mimicking executions in the reduced game to mimicking strategies, we need to formalise the relation between paths through nodes belonging to player ◯ and partial strategies for ◯. Given a (possibly infinite) path $\pi = s_1 s_2 \ldots$ where all nodes, except the last node (if it exists), belong to ◯, the partial strategy σ_π *induced by* π is such that $\sigma_\pi(s_1 \ldots s_i) = s_{i+1}$ for all $i < |\pi|$. Conversely, given a node s_1 belonging to ◯ and a strategy σ for ◯ such that $\sigma(s_i) = s_{i+1}$, the *induced path* is the longest sequence $\pi = s_1 s_2 \ldots$ such that $\mathcal{P}(s_i) = ◯$ for all $i < |\pi|$.

We are now ready to present our main correctness result. The overarching reasoning in its proof is also used in [9] to create parity game equivalence relations that preserve the winning player.

Theorem 1 *If G_r has a finite state space 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 We first give a short outline of the proof. Let s be an arbitrary node in G_r and let ◯ be the player that wins s in G, with σ as its winning strategy. To show that s is also won by ◯ in G_r, we need to construct a strategy σ_r in G_r and show that it is indeed a winning strategy for ◯ in s. The latter is achieved by showing that for any G_r-play π_r starting in s and consistent with σ_r, there is a stutter equivalent G-play π starting in s and consistent with σ. Since σ is a winning strategy for ◯, π must be won by ◯ and, by Lemma 1, so must π_r. We can then conclude that all G_r-plays starting from s and consistent with σ_r are won by ◯, and hence σ_r is a winning strategy for ◯ in s. This yields the final result that ◯ also wins s in G_r.

Let $\pi_s = s_0 s_1 \ldots$ be the path induced by some node $s \in G$ and σ, where $s = s_0$. If π_s is infinite, then the token stays in nodes owned by ◯. We apply Lemma 6 to obtain a path ρ_s in G_r and define σ_s to be the partial strategy induced by ρ_s.

Otherwise, π_s is finite and ends in a node owned by $\overline{◯}$. Here, we can apply Lemma 5 to obtain paths ρ_s in G_r and κ_s in G, such that $\pi_s \triangleq \pi_s\kappa_s \triangleq \rho_s$ and, except for the last, all nodes in ρ_s belong to ◯. This gives rise to the partial strategy σ_s that is induced by ρ_s.

Repeating this procedure for every node s owned by ◯ yields a set of partial strategies σ_s such that σ_s is defined only for the prefixes of the paths ρ_s that we constructed above. We create a total strategy σ_r in G_r by setting $\sigma_r(s_0 \ldots s_n) = \sigma_{s_i}(s_i \ldots s_n)$ if $s_i \ldots s_n$ is a prefix of ρ_{s_i}, where $s_i \ldots s_n$ is the longest non-empty suffix of $s_0 \ldots s_n$ wholly belonging to ◯. If $s_i \ldots s_n$ is not a prefix of ρ_{s_i}, then an arbitrary successor of s_n in G_r may be chosen for $\sigma_r(s_0 \ldots s_n)$.

What remains is to show that for all plays consistent with σ_r, there is a stutter equivalent play consistent with σ. Let

$\pi_r = s_0 s_1 \ldots$ be an arbitrary play in G_r and assume it is consistent with σ_r. We break up π_r into maximal subsequences $\pi_i = s_i \ldots s_j$ (respectively, $\pi = s_i \ldots$) such that $\mathcal{P}(s_i) = \ldots \mathcal{P}(s_{j-1}) \neq \mathcal{P}(s_j)$ (respectively, $\mathcal{P}(s_i) = \mathcal{P}(s_{i+1}) = \ldots$) and show how these can be transformed into subsequences π_i' to create a play π such that $\pi \triangleq \pi_r$ and π is consistent with σ. Note that there is an overlap of one node between adjacent subsequences π_i and π_{i+1}; these nodes will be preserved by our transformation.

Let $\pi_i = s_i \ldots s_j$ or $\pi = s_i \ldots$ be such a maximal subsequence. If $\mathcal{P}(s_i) = \overline{\bigcirc}$, then π_i is trivially consistent with σ, since σ only concerns $\bigcirc$-nodes. We thus choose $\pi_i' = \pi_i$.

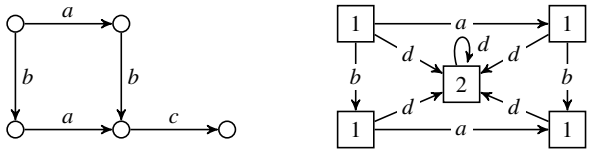
Otherwise, if $\mathcal{P}(s_i) = \bigcirc$, we use maximality of π_i to deduce that $\sigma_r(s_0 \ldots s_k) = \sigma_{s_i}(s_i \ldots s_k)$ for all k such that $i \leq k < j$, respectively, $i \leq k$ when π_i is infinite. We constructed the partial strategy σ_{s_i} from ρ_{s_i} and π_i is consistent with σ_{s_i}, so π_i must coincide with ρ_{s_i}. During the construction of ρ_{s_i} we showed that it is stutter-equivalent to π_{s_i}, which was derived from and is consistent with σ. In the infinite case, we thus choose $\pi_i' = \pi_{s_i}$. In the finite case, we have to ensure s_j is also the last node in π_i', so it can be joined up with π_{i+1}'. This is achieved by setting $\pi_i' = \pi_{s_i} \kappa_{s_i}$, which we also showed to be stutter-equivalent to ρ_{s_i}. Furthermore κ_{s_i} contains only nodes owned by $\overline{\bigcirc}$, so $\pi_{s_i} \kappa_{s_i}$ is still consistent with σ.

In all three cases, we have for each i, $\pi_i \triangleq \pi_i'$ and π_i and π_i' have the same first and, if finite, last nodes. We define π as the play $\pi_0' \pi_1' \ldots$, with the aforementioned overlap removed. Stutter-equivalence of π_r and π follows from stutter-equivalence of the subsequences. □

3.3 Optimising D2w

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

Example 5 Consider the process $(a \parallel b) \cdot c$, in which actions a and b are executed in (interleaved) parallel, and action c is executed upon termination of both a and b. In the context of this process, the μ-calculus property $\mu X.([a]X \wedge [b]X \wedge \langle - \rangle \mathit{true})$, also expressible in LTL, expresses that the action c must unavoidably be done within a finite number of steps; clearly this property holds true of the process. Below, the transition system is depicted on the left and a possible parity game encoding of our liveness property on this state space is depicted on the right. The edges in the labelled parity game that originate from the subformula $\langle - \rangle \mathit{true}$ are labelled with d.



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

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

D2t There is an event $a \in r(s)$ such that for all $a_1 \notin r(s), \ldots, a_n \notin r(s)$, if $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$, then either $s_n \xrightarrow{a}$ or there are nodes $s', s_1', \ldots, s_n'$ such that $s \xrightarrow{a} s' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n'$ and for all i, $s_i = s_i'$ or $s_i \xrightarrow{a} s_i'$.

Theorem 1 holds even for reduction functions satisfying the weak stubborn set conditions in which condition **D2t** is used instead of condition **D2w**. The proof thereof resorts to a modified construction of a mimicking winning strategy that is based on Lemma 7, described below, instead of Lemma 4.

Lemma 7 *Let r be a reduction function satisfying conditions **D1**, **D2t**, **V**, **I** and **L**. Suppose $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$ is such that $a_i \notin r(s_0)$ for every a_i occurring on this execution. Then, the following holds:*

- *If π ends in s_n, there exist a key event a_{key} and nodes $s_0', \ldots, s_n'$ such that:*
  - *$s_n \xrightarrow{a_{\mathsf{key}}} s_n'$ or $s_n = s_n'$; and*
  - *$s_0 \xrightarrow{a_{\mathsf{key}}}_r s_0' \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n'$ and $\pi \triangleq s_0 s_0' \ldots s_n'$.*
- *If π is infinite, there exists another execution $s_0 \xrightarrow{a_{\mathsf{key}}}_r s_0' \xrightarrow{a_1} s_1' \xrightarrow{a_2} \ldots$ and $\pi \triangleq s_0 s_0' s_1' \ldots$.*

Proof Let $a_{\mathsf{key}} \in r(s_0)$ be an invisible key event; its existence follows from Lemma 2 and $s_0 \xrightarrow{a_1}$ and $a_1 \notin r(s_0)$.

In case π has finite length n, we derive the existence of $s \xrightarrow{a_{\mathsf{key}}}_r s' \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n'$ either directly from **D2t** (if $s_n = s_n'$) or from **D1** (if $s_n' \xrightarrow{a_{\mathsf{key}}}$). Stutter-equivalence follows by the same reasoning as in Lemma 3.

If π is infinite, we distinguish the following cases:

- If $s \xrightarrow{a_{\mathsf{key}} a_1 \ldots a_i} s_i$ for some i, we can trivially extend this execution along π to obtain $\pi' = s \xrightarrow{a_{\mathsf{key}} a_1 \ldots a_i} s_i \xrightarrow{a_{i+1}} s_{i+1} \xrightarrow{a_{i+2}} \ldots$.
- Otherwise, if there is no i such that $s \xrightarrow{a_{\mathsf{key}} a_1 \ldots a_i} s_i$, we can apply the same reasoning as in the proof of Lemma 4.

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

We remark that the concepts of triangular and rectangular structures bear similarities to the concept of weak confluence from [12].

4 Parameterised Boolean equation systems

Parity games are used, among others, to solve *parameterised Boolean equation systems* (PBESs) [13], which, in turn, are used to answer, *e.g.*, first-order modal μ-calculus model checking problems [5]. In the remainder of this paper, we show how to apply POR in the context of solving a PBES (and, hence, the encoded decision problem). We first introduce PBESs and show how they induce labelled parity games.

A parameterised Boolean equation system is a sequence of fixed point equations over predicate formulae, *i.e.*, first-order logic formulae with second order variables. A PBES is given in the context of an abstract data type, which is used to reason about data. Non-empty data sorts of the abstract data type are typically denoted with the letters D and E. The corresponding semantic domains are $\mathbb{D}$ and $\mathbb{E}$. We assume that sorts 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 expressions 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 such an environment is denoted $[\![f]\!]\delta$. For instance, $[\![x < 2 + y]\!]\delta$ holds true iff $\delta(x) < 2 + \delta(y)$. 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 $d \neq d'$.

For conciseness, we only consider PBESs in *standard recursive form* (SRF) [29], a normal form in which each right-hand side of an equation is a *guarded* formula instead of an arbitrary (monotone) predicate formula. We remark that a PBES can be rewritten to SRF in linear time, while the number of equations grows linearly in the worst case [29, Proposition 2]. For a formalisation of the transformation to SRF, see [26, section 3.1].

Let $\mathcal{X}$ be a countable set of predicate variables. In the exposition that follows we assume for the sake of simplicity (but without loss of generality) that all predicate variables $X \in \mathcal{X}$ are of type D. We permit ourselves the use of non-uniformly typed predicate variables in our example.

Definition 6 A guarded formula ϕ is a disjunctive or conjunctive formula of the form:

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

where J is an index set, each f_j is a Boolean expression, referred to as *guard*, every e_j is a (bound) variable of sort E_j, each g_j is an expression of type D and each X_j is a predicate variable of type D. A guarded formula ϕ is said to be *total* if for each data environment δ, there is a $j \in J$ and $v \in \mathbb{E}_j$ such that $[\![f_j]\!]\delta[v/e_j]$ holds true.

We refer to the conjuncts, respectively, disjuncts, of a guarded formula as *clauses*. The denotational semantics of a guarded formula is given in the context of a data environment δ for interpreting data expressions and a *predicate environment* $\eta : \mathcal{X} \to 2^{\mathbb{D}}$, yielding an interpretation of $X_j(g_j)$ as the truth value $[\![g_j]\!]\delta \in \eta(X_j)$. Given a predicate environment and a data environment, a guarded formula induces a monotone operator on the complete lattice $(2^{\mathbb{D}}, \subseteq)$. By Tarski's theorem, least (μ) and greatest (ν) fixed points of such operators are guaranteed to exist.

Definition 7 A *parameterised Boolean equation* in SRF normal form is an equation that has the shape

$$(\mu X(d{:}D) = \phi(d)) \text{ or } (\nu X(d{:}D) = \phi(d)),$$

where $\phi(d)$ is a total guarded formula in which d is the only free data variable. A *parameterised Boolean equation system* in SRF is a sequence of parameterised Boolean equations in SRF, in which no two equations have the same left-hand side variable.

Henceforward, let

$$\mathcal{E} = (\sigma_1 X_1(d{:}D) = \varphi_1(d)) \ldots (\sigma_n X_n(d{:}D) = \varphi_n(d))$$

be a fixed, arbitrary PBES in SRF, where $\sigma_i \in \{\mu, \nu\}$. The set of *bound predicate variables* of $\mathcal{E}$, denoted by $\mathsf{bnd}(\mathcal{E})$, is the set $\{X_1, \ldots, X_n\}$. If the predicate variables occurring in the guarded formulae $\varphi_i(d)$ of $\mathcal{E}$ are taken from $\mathsf{bnd}(\mathcal{E})$, then $\mathcal{E}$ is said to be *closed*; we only consider closed PBESs. Every bound predicate variable is assigned a *rank*, where $\mathsf{rank}_{\mathcal{E}}(X_i)$ is the number of alternations in the sequence of fixpoint symbols $\nu\sigma_1\sigma_2 \ldots \sigma_i$. The sequence of fixpoint symbols in this definition is prepended with a ν to ensure that $\mathsf{rank}_{\mathcal{E}}(X_i)$ is *even* if and only if $\sigma_i = \nu$. We use the function $\mathsf{op}_{\mathcal{E}} : \mathsf{bnd}(\mathcal{E}) \to \{\vee, \wedge\}$ to indicate for each predicate variable in $\mathcal{E}$ whether the associated equation is disjunctive or conjunctive. As a notational convenience, we write J_i to refer to the index set of the guarded formula $\varphi_i(d)$, and we assume that the index sets are disjoint for different equations.

Note that for the sake of simplicity, we only consider PBESs in which each equation carries the same parameter $d{:}D$. The theory presented in the following can be straightforwardly extended to support PBESs where each equation has a different number of parameters, something we also permit ourselves in the examples.

4.1 Semantics

The standard denotational fixed point semantics of a closed PBES associates a subset of $\mathbb{D}$ to each bound predicate variable (*i.e.*, their meaning is independent of the predicate environment used to interpret guarded formulae). For details of the standard denotational fixed point semantics of a PBES we refer to [13]. We forego the denotational semantics and instead focus on the (provably equivalent, see *e.g.* [6,29]) game semantics of a PBES in SRF. Below, recall that, in the PBES $\mathcal{E}$ that we fixed, each predicate variable X_i carries a data parameter named d and uses the set J_i to index the clauses in its right-hand side.

Definition 8 The *solution* to $\mathcal{E}$ is a mapping $[\![\mathcal{E}]\!] : \mathsf{bnd}(\mathcal{E}) \to 2^{\mathbb{D}}$, defined as

$$[\![\mathcal{E}]\!](X_i) = \{v \in \mathbb{D} \mid (X_i, v) \text{ is won by } \Diamond \text{ in } G_{\mathcal{E}}\},$$

where $X_i \in \mathsf{bnd}(\mathcal{E})$ and $G_{\mathcal{E}}$ is the parity game associated with $\mathcal{E}$. The game $G_{\mathcal{E}} = (V, E, \Omega, \mathcal{P})$ is defined as:

- $V = \mathsf{bnd}(\mathcal{E}) \times \mathbb{D}$ is the set of nodes;
- E is the edge relation, satisfying $(X_i, v) \to (X_j, w)$ for given X_i, $j \in J_i$, v and w if and only if there exists an environment δ such that both $[\![f_j]\!]\delta[v/d]$ and $w = [\![g_j]\!]\delta[v/d]$ hold;
- $\Omega((X_i, v)) = \mathsf{rank}_{\mathcal{E}}(X_i)$; and
- $\mathcal{P}((X_i, v)) = \Diamond$ iff $\mathsf{op}_{\mathcal{E}}(X_i) = \vee$.

In the definition of the transition relation E, the successors of (X_i, v) are exactly those (X_j, w) which may influence the validity of the right-hand side φ_i of X_i; this does not depend on whether φ_i is disjunctive or conjunctive. Note that the parity game $G_{\mathcal{E}}$ may have an infinite state space when $\mathbb{D}$ is infinite. In practice, we are often interested in the part of the parity game that is reachable from some initial node (X, v); this is often (but not always) finite. We illustrate the PBES semantics with an example.

Example 6 Consider the following PBES $\mathcal{E}$ in SRF:

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

The six nodes in the parity game $G_{\mathcal{E}}$ which are reachable from (X, true) are depicted in Fig. 8. We study the clause $\exists n{:}N.n \leq 2 \wedge Y(b, n)$ in detail. The conditions $[\![n \leq 2]\!]\delta[\mathit{true}/b]$ and $\mathit{true}, 2 = [\![b, n]\!]\delta[\mathit{true}/b]$ from Definition 8 are satisfied if $\delta(n) = 2$. Hence, the edge $(X, \mathit{true}) \to$ $(Y, \mathit{true}, 2)$ is present in the game. In fact, all three horizontal edges at the top stem from this same clause.

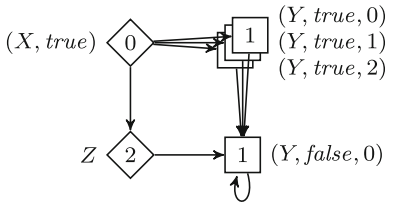


Fig. 8 Reachable part of the parity game underlying the PBES of Example 6, when starting from node (X, true)

The leftmost vertical edge originates from the clause $b \wedge Z$, while the horizontal edge from Z to $(Y, \mathit{false}, 0)$ originates from $\mathit{true} \wedge Y(\mathit{false}, 0)$. The remaining edges, including the selfloop, are induced by the clause $\mathit{true} \Rightarrow Y(\mathit{false}, 0)$.

All plays in this game end in the node $(Y, \mathit{false}, 0)$ and iterate there ad infinitum. Since its priority is odd, all plays – and consequently all nodes – are won by player odd. This means that the semantics of $\mathcal{E}$ satisfies $\mathit{true} \notin [\![\mathcal{E}]\!](X)$. □

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

Definition 9 Let $G_{\mathcal{E}}$ be the parity game associated with $\mathcal{E}$. The labelled parity game associated with $\mathcal{E}$ is the structure $(G_{\mathcal{E}}, \mathsf{evt}(\mathcal{E}), \ell)$, where $G_{\mathcal{E}}$ is as defined in Definition 8, and, for $j \in J_i$, we have

$$\ell(j) = \{\langle (X_i, v), (X_j, w)\rangle \in E \mid [\![f_j]\!]\delta[v/d] \text{ holds true and } w = [\![g_j]\!]\delta[v/d] \text{ for some } \delta\}$$

It follows from the definition that the event $j \in J_i$ is *invisible* if $\mathsf{rank}_{\mathcal{E}}(X_i) = \mathsf{rank}_{\mathcal{E}}(X_j)$ and $\mathsf{op}_{\mathcal{E}}(X_i) = \mathsf{op}_{\mathcal{E}}(X_j)$, and *visible* otherwise. Since solving the parity game G corresponding to a PBES $\mathcal{E}$ also yields a solution for $\mathcal{E}$ (Definition 8) and POR preserves the solution of a parity game (Theorem 1), it suffices to solve a reduced game G_r to obtain a (partial) solution to $\mathcal{E}$.

As discussed before, our main application domain is model checking. When checking a μ-calculus formula φ on a transition system T, we typically only need to determine whether the initial node of T satisfies φ. That corresponds to finding the solution of a single node (X, v) in the PBES that encodes T and φ. Our approach is thus, given a PBES $\mathcal{E}$, to compute a reduced game G_r, starting from (X, v), and then determine the winner of (X, v) in G_r. If G_r is sufficiently smaller than the full game G and the overhead of computing the reduction function r is limited, this is faster than solving the PBES through the full game G.

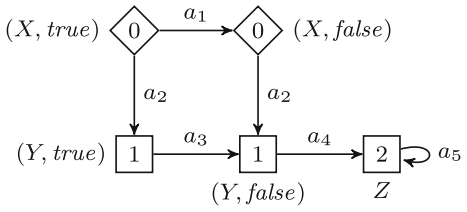


Fig. 9 Labelled parity game corresponding to the PBES of Example 7

4.2 Choice of edge labelling

The labelled parity game derived from a PBES $\mathcal{E}$ associates exactly one event with every clause in $\mathcal{E}$. However, our POR correctness theorem (Theorem 1) in principle allows any edge labelling as long as the stubborn set conditions are satisfied. The choice for the labelling function does influence the amount of reduction that can be achieved. In the extreme cases that every edge has a unique label or all edges are labelled the same, no reduction can be achieved. The below example shows a case for which our current labelling function hinders reduction.

Example 7 Consider the SRF-PBES below, where the name of each clause is indicated on the right, and its labelled parity game in Fig. 9.

$$\begin{aligned} \nu X(b{:}B) &= b \wedge X(\mathit{false}) && (a_1)\\ &\quad \vee Y(b) && (a_2)\\ \mu Y(b{:}B) &= b \Rightarrow Y(\mathit{false}) && (a_3)\\ &\quad \wedge \neg b \Rightarrow Z && (a_4)\\ \nu Z &= Z && (a_5) \end{aligned}$$

Note that a_2 and a_4 are visible; the other events are invisible. It is impossible to reduce the parity game: the only viable location is (X, true), but it does not allow stubborn sets smaller than $\{a_1, a_2, a_4\}$. Setting $r((X, \mathit{true})) = \{a_1\}$ is not possible due to **D2w**, since a_1 is not a key event in that case. Conversely, $r((X, \mathit{true})) = \{a_2\}$ is not allowed by **D1**. Furthermore, if we have $a_2 \in r((X, \mathit{true}))$, then **V** requires that $a_4 \in r((X, \mathit{true}))$. However, if we label both the a_1 edge and the a_3 edge with a, then it becomes possible to set $r((X, \mathit{true})) = \{a\}$. □

The example suggests that it can be beneficial to use the same event for multiple clauses. Specifying which clauses should correspond to the same event can be done with an equivalence relation on clauses. The next definition formalises the parity game that follows from such an equivalence relation.

Definition 10 Let $\mathcal{E}$ be an SRF-PBES, $\mathcal{R} \subseteq \mathsf{evt}(\mathcal{E}) \times \mathsf{evt}(\mathcal{E})$ an equivalence relation on events and ${}^{\mathsf{evt}(\mathcal{E})}/_{\mathcal{R}}$ the corresponding set of equivalence classes. Then, the labelled parity game corresponding to $\mathcal{E}$ and $\mathcal{R}$ is the structure $(G_{\mathcal{E}}, {}^{\mathsf{evt}(\mathcal{E})}/_{\mathcal{R}}, \ell)$, where $G_{\mathcal{E}}$ is the parity game corresponding to $\mathcal{E}$, and, for all $X_i \in \mathsf{bnd}(\mathcal{E})$ and $J \in {}^{\mathsf{evt}(\mathcal{E})}/_{\mathcal{R}}$, the labelling $\ell(J)$ is defined as follows:

$$\ell(J) = \bigcup_{X_i \in \mathsf{bnd}(\mathcal{E}),\, j \in J_i \cap J} \{((X_i, v), (X_j, w)) \mid [\![f_j]\!]\delta[v/d] \text{ holds true and } w = [\![g_j]\!]\delta[v/d] \text{ for some } \delta\}$$

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

5 PBES solving using POR

A consequence of the partial-order reduction theorem is that a reduced parity game suffices for computing the truth value to $X(e)$ for a given PBES $\mathcal{E}$ with $X \in \mathsf{bnd}(\mathcal{E})$. However, **D1**, **D2w/D2t** and **L** are conditions on the (reduced) state space as a whole and, hence, hard to check locally. We therefore approximate these conditions in such a way that we can construct a stubborn set while generating the state space.

From hereon, let $\mathcal{E}$ again be a PBES in SRF with equations of the shape $\sigma_i X_i(d{:}D) = \varphi_i$ and $(G, \mathcal{A}, \ell)$, with $G = (V, E, \Omega, \mathcal{P})$, its labelled parity game. The most common local condition for **L** is the *stack proviso* $\mathbf{L}^S$ [32]. This proviso assumes that the state space is explored with *depth-first search* (DFS), and it uses the *Stack* that stores unexplored nodes. Any transition leading back from the current node to a node in the DFS stack closes a cycle. In that case, the node must be *fully expanded*, *i.e.*, $r(s) = \mathcal{A}$. The following should thus hold for all nodes $s \in V_r$ encountered during exploration:

$\mathbf{L}^S$ If $\mathit{succ}_{G_r}(s) \cap \mathit{Stack} \neq \emptyset$, then $r(s) = \mathcal{A}$.

We use the *color proviso* [8], which improves on the above condition with some additional bookkeeping of which nodes on the stack are or will be fully expanded. As a result, not every transition that leads back to the DFS stack requires its originating node to be fully expanded, improving reduction potential.

Locally approximating conditions **D1** and **D2w** requires a static analysis of the PBES. For this, we draw upon ideas from [22] and extend these to properly deal with non-determinism. In the following definitions, any relation on the set $\mathcal{A}$ of events is also a relation on equivalence classes of clauses in the PBES, see Definition 10. Furthermore, if the event a corresponds to the clauses that contain condition f_a, statements

such as "for all $s \in V$, it holds $s \xrightarrow{a}$" can be approximated on the PBES level by checking whether the event a occurs in each right-hand side of the PBES and evaluating the expression $\forall d{:}D.\exists e_a{:}E_a.f_a$, where d is the data parameter carried by each equation in our PBES $\mathcal{E}$. Whether this closed Boolean expression is equal to *true* or equal to *false* can, for example, be determined with the help of an SMT-solver.

To reason about which events are independent, we rely on the idea of *accordance*.

Definition 11 Let $a, b \in \mathcal{A}$. We define the *accordance* relations *DNL*, *DNS*, *DNT* and *DNA* on $\mathcal{A}$ as follows:

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

Note that *DNL* and *DNT* are not necessarily symmetric relations. An illustration of the conditions for left-according, square-according and triangle-according is given in Fig. 10.

For the case of left-accordance, we illustrate how the definition can be translated to an analysis on the level of PBESs. Recall that a clause $\exists e_j{:}E_j.f_j \wedge X_j(g_j)$ (respectively $\forall e_j{:}E_j.f_j \Rightarrow X_j(g_j)$) contains a condition f_j as well as an update expression g_j that determines what the next state is. The expression f_j, respectively, g_j, depends on the data parameter d of the surrounding equation, as well as on the quantified variable e_j. We write $f_j(d, e_j)$, respectively, $g_j(d, e_j)$, to make this explicit. Given an event a (or, equivalently, equivalence class of clauses), we write $X_i \overset{a}{\hookrightarrow} Y$ if and only if there is a $j \in a \cap J_i$ such that $Y = X_j$, *i.e.*, a clause $j \in a$ occurs in the right-hand side of X_i and leads to Y. With this, left-accordance of a with b can be characterised exactly by

$$\begin{aligned}
&\forall d, e_b, e_a, X, X_1, X' \in \mathsf{bnd}(\mathcal{E}).(f_b(d, e_b) \wedge \\
&\quad f_a(g_b(d, e_b), e_a) \wedge X \overset{b}{\hookrightarrow} X_1 \wedge X_1 \overset{a}{\hookrightarrow} X') \Rightarrow \\
&\quad (\exists e'_a, e'_b, X_2 \in \mathsf{bnd}(\mathcal{E}).f_a(d, e'_a) \wedge f_b(g_a(d, e'_a), e'_b) \wedge \\
&\qquad g_a(g_b(d, e'_b), e'_a) = g_b(g_a(d, e'_a), e'_b) \wedge \\
&\qquad\quad X \overset{a}{\hookrightarrow} X_2 \wedge X_2 \overset{b}{\hookrightarrow} X')
\end{aligned}$$

This expression considers whether the left-accordance condition is satisfied by the data expressions f_a, f_b, g_a and g_b contained in $\mathcal{E}$, as well as by the presence of clauses in different equations (as captured by $\overset{a}{\hookrightarrow}$ and $\overset{b}{\hookrightarrow}$).

Accordance relations safely approximate the independence of events. The dependence of events, required for satisfying **D2w**, can be approximated using Godefroid's *necessary enabling sets* [11].

Definition 12 Let a be an event that is disabled in some node s. A *necessary-enabling set* (NES) for a in s is any set $NES_s(a) \subseteq \mathcal{A}$ such that for every execution $s \xrightarrow{a_1 \ldots a_n a}$ there is at least one a_i such that $a_i \in NES_s(a)$.

For every node and event there might be more than one NES. In particular, every superset of a NES is also a NES. A larger-than-needed NES may, however, have a negative impact on the reduction that can be achieved. Given a PBES, computing a NES can be done in a similar fashion as the accordance relations, by constructing an expression over elements of the clauses and analysing their presence in each equation.

The following lemmata show how the accordance relations and necessary-enabling set can be used to implement conditions **D1**, **D2w** and **D2t**, respectively. A combination of Lemmata 8 and 9 in a deterministic setting appeared as Lemma 1 in [22]. Note that as a notational convention we write $R(a)$ to denote the projection $\{b \mid (a, b) \in R\}$ of a binary relation.

Lemma 8 *A reduction function r satisfies **D1** in node $s \in V$ if for all $a \in r(s)$:*

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

Proof Let s be an arbitrary node and let r be a reduction function that satisfies the conditions above. Furthermore, let $s \xrightarrow{a_1 \ldots a_n a} s'_n$ be an execution such that $a_1 \notin r(s), \ldots, a_n \notin r(s)$ and $a \in r(s)$. We distinguish the following cases:

- If a is disabled in s, it must hold that $NES_s(a) \subseteq r(s)$ for some NES_s. However, according to the definition of a necessary-enabling set, at least one of $a_1, \ldots, a_n$ is contained in $NES_s(a)$ and thus in $r(s)$. Since this contradicts our assumption that $a_1 \notin r(s), \ldots, a_n \notin r(s)$, we conclude that the execution $s \xrightarrow{a_1 \ldots a_n a} s'_n$ does not exist, and so **D1** is satisfied.
- If a is enabled in s, it must be that $DNL(a) \subseteq r(s)$. Since that implies $a_1, \ldots, a_n \notin DNL(a)$, it follows that for every a_i and all nodes t, t_1 and t', the following holds:

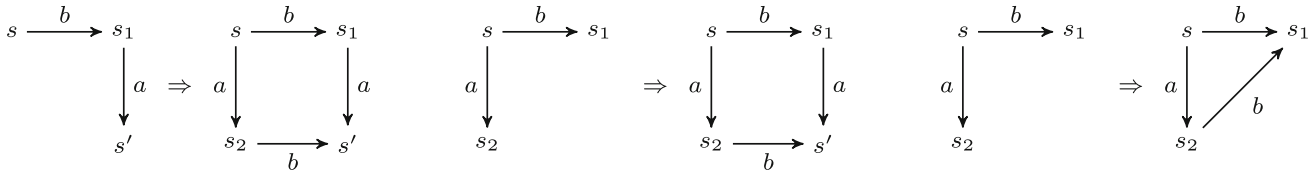
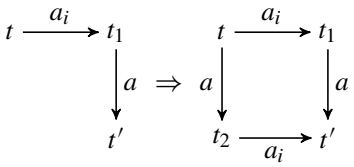
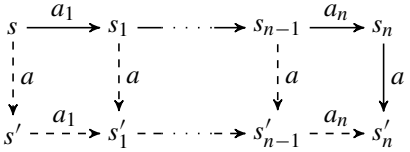


Fig. 10 Illustrations of the concepts left-according, square-according and triangle-according



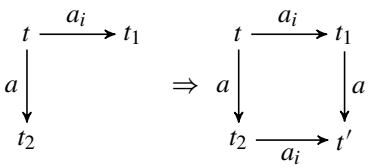
By inductively applying this implication from right to left on the execution $s \xrightarrow{a_1 \ldots a_n} s_n \xrightarrow{a} s'_n$, we derive the existence of the dashed transitions in the figure below.



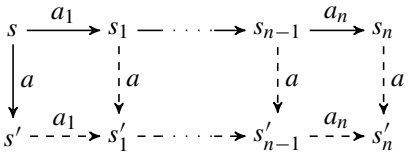
We conclude that the conditions of **D1** are satisfied. □

Lemma 9 *A reduction function r satisfies **D2w** in a node $s \in V$ if there is an enabled event $a \in r(s)$ such that $DNS(a) \subseteq r(s)$.*

Proof Let $s \xrightarrow{a_1 \ldots a_n} s_n$ be an execution such that all $a_1, \ldots, a_n \notin r(s)$ and let $a \in r(s) \cap enabled(s)$ be an event such that $DNS(a) \subseteq r(s)$. We deduce that $a_1, \ldots, a_n \notin DNS(a)$, and thus the following implication holds for all a_i and nodes t, t_1 and t_2:



Applying this inductively from left to right on the transition $s \xrightarrow{a} s'$ and the execution $s \xrightarrow{a_1 \ldots a_n} s_n$, we derive the existence of the dashed transitions in the following figure.



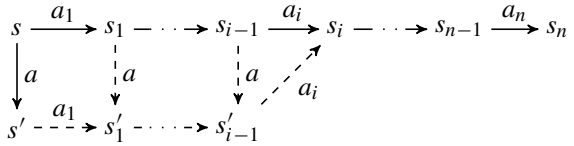
Hence, a satisfies the conditions of **D2w**. □

Lemma 10 *A reduction function r satisfies **D2t** in a node s if there is an enabled event $a \in r(s)$ such that $DNA(a) \subseteq r(s)$.*

Proof Let $s \xrightarrow{a_1 \ldots a_n} s_n$ be an execution such that all $a_1, \ldots, a_n \notin r(s)$ and let $a \in r(s) \cap enabled(s)$ be an event such that $DNA(a) \subseteq r(s)$. We distinguish two cases:

- It holds that $a_1, \ldots, a_n \in DNT(a)$. Since $a_1 \notin r(s), \ldots, a_n \notin r(s)$ and $r(s) \supseteq DNA(a) = DNS(a) \cap DNT(a)$, we can deduce that $a_1, \ldots, a_n \notin DNS(a)$. By following the same reasoning as in the proof of Lemma 9, we derive the validity of **D2t**.
- There is an $0 < i \leq n$ such that $a_i \notin DNT(a)$. We consider the smallest such i, *i.e.*, $a_1, \ldots, a_{i-1} \in DNT(a)$. With $a_1 \notin r(s), \ldots, a_n \notin r(s)$ and $r(s) \supseteq DNA(a) = DNS(a) \cap DNT(a)$, we deduce that $a_1, \ldots, a_{i-1} \notin DNS(a)$ and $a_i \notin DNT(a)$.

By first applying the square-according relation from left to right on the events a and $a_1, \ldots, a_{i-1}$ and then applying the triangle-according relation on a and a_i, we derive the existence of the dashed transitions in the following figure.



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

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

$$det(a) \text{ iff } [\![f_a]\!]\delta \text{ and } [\![f_a]\!]\delta' \text{ implies } [\![g_a]\!]\delta = [\![g_a]\!]\delta' \quad \text{for all } \delta, \delta' \text{ with } \delta(d) = \delta'(d).$$

The following lemma specialises Lemma 8 and shows how knowledge of deterministic events can be applied to potentially improve the reduction.

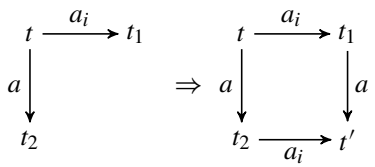
Lemma 11 *A reduction function r satisfies **D1** in a node s if for all $a \in r(s)$:*

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

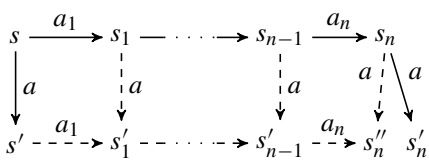
– *if* $det(a)$ *and* a *is enabled in* s*, then* $DNS(a) \subseteq r(s)$ *or* $DNL(a) \subseteq r(s)$.
– *if* $\neg det(a)$ *and* a *is enabled in* s*, then* $DNL(a) \subseteq r(s)$.

Proof Let s be an arbitrary node and let r be a reduction function that satisfies the conditions above. For the cases where a is disabled or a is enabled and $DNL(a) \subseteq r(s)$, see the proof of Lemma 8. Here, we only consider the new case where a is deterministic and enabled in s and $DNS(a) \subseteq r(s)$. □

Let $s \xrightarrow{a_1 \dots a_n} s_n \xrightarrow{a} s'_n$ be an execution such that $a_1 \notin r(s), \dots, a_n \notin r(s)$ and $a \in r(s)$ and let $s \xrightarrow{a} s'$. The following implication holds for all a_i and nodes t, t_1 and t_2:



Applying this inductively from left to right on the transition $s_n \xrightarrow{a} s'_n$ and the execution $s \xrightarrow{a_1 \dots a_n} s_n$, we deduce the existence of the dashed transitions for some node s''_n.



Since a is deterministic it follows that $s'_n = s''_n$. Regardless of whether a is invisible, the transitions $s_i \xrightarrow{a} s'_i$ are present for every $1 \leq i < n$, and thus **D1** is satisfied. □

Since relations *DNS* and *DNL* are incomparable we cannot decide *a priori* which should be used for deterministic events. However, Lemma 11 permits the choice between *DNS* and *DNL* to be made individually for every state, so we can do this during exploration of the state space. This choice can be made based on a heuristic function, similar to the function for NESs proposed in [22].

As sketched above, both the accordance relations and the NESs can be computed based on the condition f_a and update expression g_a associated with each clause, as well as the presence of clauses captured by $\overset{a}{\hookrightarrow}$. Evaluation of the expressions constructed in this way can be computationally expensive, so often it is better to compute a (cheaper) approximation. In a PBES with multiple parameters per predicate variable, this can, for example, be done by analysing which parameters influence the validity of condition f_a and which parameters are changed in the update functions g_a Two clauses a and b that operate on disjoint sets of parameters are often left- and square-according (depending on $\overset{a}{\hookrightarrow}$ and $\overset{b}{\hookrightarrow}$). NESs may be approximated using techniques to extract control flow graphs from a PBES [19], although we have not done so.

6 Experiments

We implemented the ideas from the previous section in a prototype tool, called `pbespor`, as part of the mCRL2 toolset [5]; it is written in C++. Our tool converts a given input PBES to a PBES in SRF, runs a static analysis to compute the accordance relations (see Sect. 5), and uses a depth-first exploration to compute the parity game underlying the PBES in SRF. Since our previous work [30], we have updated the implementation to produce stubborn sets that adhere to the new condition **P**. The static analysis relies on an external SMT solver. (We use Z3 in our experiments.) Experiments are conducted on a machine with four Intel Xeon 6136 CPUs @ 3.00GHz and 3TB of RAM, running mCRL2 with Git commit hash `b7976da39c4e`. All our code is single-threaded, so similar performance may be obtained on a more modest machine.

To measure the effectiveness of our implementation, we analysed the following models, encoded in mCRL2[4]: Anderson's mutual exclusion protocol [1], the dining philosophers problem, the gas station problem [15], Hesselink's handshake register [16], Le Lann's leader election protocol [23], Milner's Scheduler [25] and the Krebs cycle of ATP production in biological cells (model inspired by [31]). Most of these models are scalable, *i.e.*, the number of components can easily be adjusted. Each model is subjected to one or more requirements phrased as mCRL2's first-order modal μ-calculus formulae. If possible, Table 1 provides a CTL* formula that captures the essence of the requirement.

We analyse the effectiveness of our partial-order reduction technique by measuring the reduction of the size of the state space, and the time that is required to generate the state space. Since the static analysis that is conducted can require a non-negligible amount of time, we pay close attention to the various forms of static analysis that can be conducted. In particular, we compare the total time and effectiveness (in terms of reduction) of running the following static analysis:

– computing left-accordance (the relation *DNL*) vs. over-approximating it with the complete relation.
– computing a NES vs. over-approximating it with the set of all events $\mathcal{A}$.
– using **D2w** vs. the use of **D2t** (*i.e.*, use Lemma 9 vs. Lemma 10);

[4] The models are archived online at https://doi.org/10.5281/zenodo.3602969.

As a baseline for comparisons, we take a basic static analysis (over-approximated DNL, over-approximated NES, **D2w**), see column "basic" in Table 1. In order to guarantee termination of the static analysis phase, we set a timeout of 200 ms per formula that is sent to the solver. Table 1 reports on the statistics we obtained for exploring the full state space and the four possible POR configurations described above; the table is sorted with respect to the time needed for a full exploration. The time we list consists of the time needed to conduct the analysis plus the time needed for the exploration.

For most small instances, the time required for static analysis dominates any speed-up gained by the state space reduction. When the state spaces are larger, it becomes more likely to achieve a speed-up, while the highest overhead suffered by "basic" is 55% (Hesselink, cache consistency). Significant reduction can be achieved even for non-trivial properties, such as "lann.5" with "no data loss". Scheduler is an extreme case: its processes have very few dependencies, leading to an exponential reduction, both in terms of the state space size and in terms of time. In several cases, the use of a NES or **D2t** brings extra reduction (highlighted in bold). Moreover, the extra time required to conduct the additional analysis seems limited. The use of DNL, on the other hand, never pays off in our experiments; it even results in a slightly larger state space in two cases. These results are very similar to those of our earlier experiments [30], and thus, the addition of condition **P** has little impact on the reduction for these particular PBESs.

We note that there are also models, not listed in Table 1, where our static analysis does not yield any useful results and no reduction is achieved. Even if in such cases a reduction would be possible in theory, the current static analysis engines are unable to deal with the more complex data types often used in such models; *e.g.*, recursively defined lists or infinite sets, represented symbolically with higher-order constructions. This calls for further investigations into static analysis theories that can effectively deal with complex data.

Furthermore, we remark that all the models we listed here contain *symmetry*: two or more processes that show almost the same behaviour. This may mean that our results cannot be generalised to non-symmetric models, although [7] suggests that partial-order reduction and symmetry reduction exploit different aspects of a model.

Finally, we point out that in the case of, *e.g.*, the dining philosophers problem, the relative reduction under the "no deadlock" property is much better than under the "$\forall\Box\forall\Diamond eat$" property. This demonstrates the impact properties can have on the reductions achievable and the importance of the way edges are labelled. We explain this in the following example.

Example 8 Consider the PBES below, which encodes the formula $\nu X.([-]X \wedge \forall i.\mu Y.([a_{1-i}]Y \wedge \langle - \rangle true))$ on the transition system of Fig. 11a. For reference, in Table 1, we denoted formulae of this shape as $\forall\Box\forall\Diamond a_i$. Below, the names of each of the clauses are indicated on the right.

$$\begin{aligned}
\nu X(b_0, b_1{:}B) &= b_0 \Rightarrow X(false, b_1) && (j_0)\\
&\wedge\ b_1 \Rightarrow X(b_0, false) && (j_1)\\
&\wedge\ \forall b{:}B.Y(b_0, b_1, b) && (xy)\\
\mu Y(b_0, b_1, b{:}B) &= (b \wedge b_0) \Rightarrow Y(false, b_1, b) && (j_0')\\
&\wedge\ (\neg b \wedge b_1) \Rightarrow Y(b_0, false, b) && (j_1')\\
&\wedge\ \neg(b \wedge b_0 \vee \neg b \wedge b_1) \Rightarrow Z && (f)\\
\mu Z &= Z && (f')
\end{aligned}$$

The event xy represents the transition from fixpoint X into Y, which does not involve an action from the transition system. Note that the complete state space is encoded once in the fixpoint X and twice in Y, albeit with a subset of the transitions. In the corresponding labelled parity game, depicted in Fig. 11b, no reduction can be achieved. □

To achieve reduction in this example, we need to do three things. First, the quantifier in the clause xy needs to be unfolded; this yields two clauses, namely $Y(b_0, b_1, false)$ (name xy_0) and $Y(b_0, b_1, true)$ (name xy_1). Furthermore, we should identify clauses j_0 and j_0' (resp. j_1 and j_1'); the resulting event is called a_0 (resp. a_1). Lastly, we have to change the fixpoint of X to ensure xy_0 and xy_1 are invisible. Remark that this does not change the solution of the PBES. In that case, four nodes can be eliminated, see Fig. 11c.

In the experiments, we achieve the identification of clauses j_0 and j_0' (resp. j_1 and j_1') by partially *instantiating* the PBES [34]. This procedure is implemented in the mCRL2 tool `pbesinst`. In the example above, instantiating parameter $b{:}B$ of Y results in two new equations:

$$\begin{aligned}
\mu Y_{true}(b_0, b_1) &= b_0 \Rightarrow Y_{true}(false, b_1) && (j_0)\\
&\wedge\ \neg b_0 \Rightarrow Z\\
\mu Y_{false}(b_0, b_1) &= b_1 \Rightarrow Y_{false}(b_0, false) && (j_1)\\
&\wedge\ \neg b_1 \Rightarrow Z
\end{aligned}$$

The clauses in these equations can be identified with the first two clauses in the right-hand side of X with our original strategy based on syntactic equivalence of the condition and update expressions.

7 Conclusion

We have presented an approach for applying partial-order reduction on parity games. This has two main advantages over POR applied on labelled transition systems or Kripke structures: our approach supports the full modal μ-calculus, not just a fragment thereof, and the potential for reduction is greater, because we do not require a singleton proviso.

Table 1 Runtime (analysis + exploration; in seconds) and number of states when exploring either the full state space or the reduced state space, for four different static analysis approaches

Model	Property	Full		Basic		+DNL		+NES		+D2t	
		Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time	Nodes	Time
gas station.c3	$\exists\Box\exists\Diamond pumping$	1261	0.11	967	0.89	967	2.20	967	1.77	967	1.55
gas station.c3	No deadlock	1197	0.13	735	0.79	735	2.17	735	1.69	735	1.27
gas station.c3	$\exists\Diamond accept$	1197	0.15	1077	0.83	1077	2.08	1077	1.55	**735**	1.36
scheduler8	No deadlock	3073	0.28	34	0.18	34	0.64	34	0.45	34	0.30
scheduler10	No deadlock	15361	1.70	42	0.23	42	0.77	42	0.59	42	0.36
anderson.5	$\forall\Diamond cs$	23597	4.44	2 957	2.63	2957	5.56	2957	3.44	2957	4.26
hesselink	Cache consistent	91,009	5.66	82602	7.39	83,601	10.45	81,673	8.18	**71,943**	7.49
dining10	No deadlock	154,451	19.89	4743	0.82	4743	1.59	4743	1.37	4743	1.00
krebs.3	$\forall\Diamond energy$	238,877	24.17	232,273	25.38	232,273	25.38	**209,345**	22.13	232,273	25.72
gas station.c6	$\exists\Box\exists\Diamond pumping$	192,700	35.52	114,130	23.12	114,130	26.93	114,130	25.82	114,130	24.96
gas station.c6	$\exists\Diamond accept$	186,381	38.65	150,741	35.63	150,741	41.01	150,741	39.22	**75,411**	19.56
gas station.c6	No deadlock	186,381	47.52	75,411	21.41	75,411	24.11	75,411	24.51	75,411	23.09
scheduler14	No deadlock	344,065	58.11	58	0.37	58	1.20	58	0.94	58	0.55
hesselink	$\forall\Box(wr \Rightarrow \exists\Diamond fin)$	1,047,233	64.91	1,013,441	75.54	1,013,441	77.77	1,013,441	75.96	**791,270**	54.74
hesselink	$\forall\Box(wr \Rightarrow \forall\Diamond fin)$	1,047,232	70.82	791,371	59.66	791,347	61.25	**749,212**	58.82	791,339	59.21
lann.5	Consistent data	818104	115.27	818,104	135.95	818,104	145.34	818,104	146.90	**761,239**	124.96
krebs.4	$\forall\Diamond energy$	1,047,406	136.06	971,128	123.41	971,128	122.04	**843,349**	104.05	971,128	120.08
anderson.5	No deadlock	689,901	142.89	258,272	65.02	257,821	68.06	257,993	77.80	257,921	67.93
lann.5	No data loss	1,286,452	182.16	453,130	64.00	453,130	68.91	453,130	67.42	453,130	64.17
dining10	$\forall\Box\forall\Diamond eat$	1,698,951	228.50	100,619	11.77	100,680	12.30	100,763	12.23	100,580	11.57
anderson.7	$\forall\Diamond cs$	3,964,599	1397.89	124,707	63.95	124,707	74.11	124,707	68.74	124,707	70.27
scheduler18	No deadlock	7,077,889	1520.15	74	0.54	74	1.73	74	1.31	74	0.82

Figures printed in boldface indicate which of the additional static analyses is able to achieve the largest reduction over "basic" (if any)

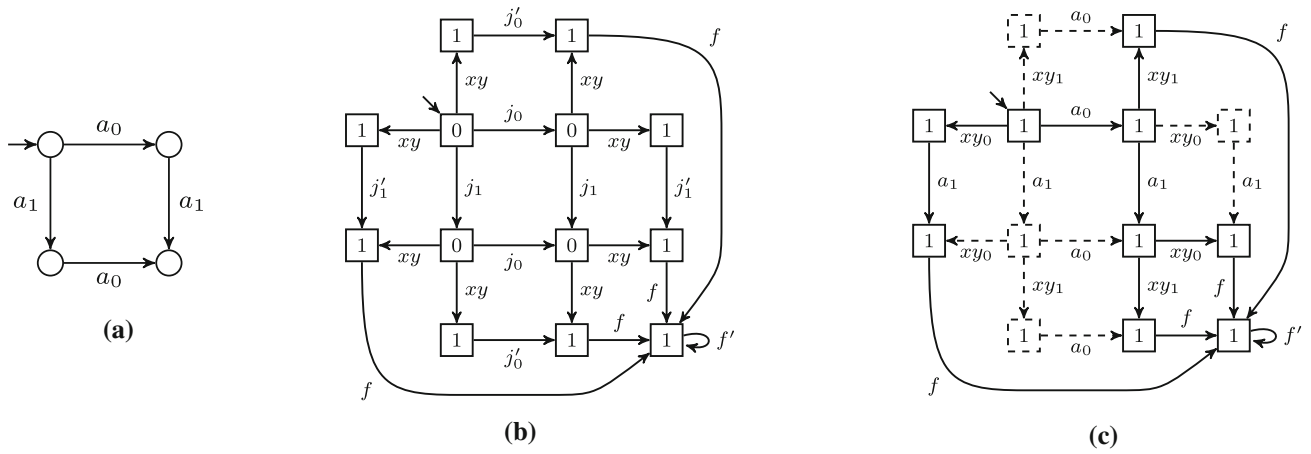


Fig. 11 A transition system and two parity games showing that, although the transition system can be reduced, it is not always possible to reduce the corresponding parity game. After tweaking the edge labelling, some reduction is possible

Furthermore, we have shown how the ideas can be implemented with PBESs as a high-level representation. In future work, we aim to gain more insight into the effect of identifying events across PBES equations in several ways. We also want to investigate the possibility of solving a reduced parity game while is it being constructed. In certain cases, one may be able to decide the winner of the original game from this partial solution.

Acknowledgements We would like to thank the reviewers for their in-depth comments. Their suggestions helped us improve the presentation significantly.

References

1. Anderson, T.E.: The performance of spin lock alternatives for shared-memory multiprocessors. IEEE Trans. Parallel Distrib. Syst. **1**(1), 6–16 (1990). https://doi.org/10.1109/71.80120
2. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Bønneland, F.M., Jensen, P.G., Larsen, K.G., Muñiz, M.: Partial order reduction for reachability games. In: CONCUR 2019, **140**, 23:1–23:15 (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.23
4. Bønneland, F.M., Jensen, P.G., Larsen, K.G., Muñiz, M., Srba, J.: Stubborn set reduction for two-player reachability games. Log. Methods Comput. Sci. **17**(1), 1–26 (2021). https://doi.org/10.23638/LMCS-17(1:21)2021
5. 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
6. 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
7. Emerson, E.A., Jha, S., Peled, D.: Combining partial order and symmetry reductions. In: TACAS 1997, LNCS, vol. 1217, pp. 19–34 (1997). https://doi.org/10.1007/BFb0035378
8. Evangelista, S., Pajault, C.: Solving the ignoring problem for partial order reduction. Int. J. Softw. Tools Technol. Transf. **12**, 155–170 (2010). https://doi.org/10.1007/s10009-010-0137-y
9. Gazda, M., Willemse, T.A.C.: On parity game preorders and the logic of matching plays. In: SOFSEM 2016, LNCS, vol. 9587, pp. 277–289 (2016). https://doi.org/10.1007/978-3-662-49192-8_23
10. Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. Inf. Comput. **150**(2), 132–152 (1999). https://doi.org/10.1006/inco.1998.2778
11. Godefroid, P.: Partial-Order Methods for the Verification of Concurrent Systems, LNCS, vol. 1032. Springer, Berlin (1996). https://doi.org/10.1007/3-540-60761-7
12. Groote, J.F., Sellink, M.P.A.: Confluence for process verification. Theor. Comput. Sci. **170**(1–2), 47–81 (1996). https://doi.org/10.1016/s0304-3975(96)00175-2
13. Groote, J.F., Willemse, T.A.C.: Parameterised Boolean equation systems. Theor. Comput. Sci. **343**(3), 332–369 (2005). https://doi.org/10.1016/j.tcs.2005.06.016
14. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Trans. Program. Lang. Syst. **16**(3), 843–871 (1994). https://doi.org/10.1145/177492.177725
15. Heimbold, D., Luckham, D.: Debugging ada tasking programs. IEEE Softw. **2**(2), 47–57 (1985). https://doi.org/10.1109/MS.1985.230351
16. Hesselink, W.H.: Invariants for the construction of a handshake register. Inf. Process. Lett. **68**(4), 173–177 (1998). https://doi.org/10.1016/s0020-0190(98)00158-6
17. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal Methods Syst. Des. **9**(1–2), 41–75 (1996). https://doi.org/10.1007/BF00625968

18. Kan, S., Huang, Z., Chen, Z., Li, W., Huang, Y.: Partial order reduction for checking LTL formulae with the next-time operator. J. Log. Comput. **27**(4), 1095–1131 (2017). https://doi.org/10.1093/logcom/exw004
19. 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
20. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. **27**(3), 333–354 (1982). https://doi.org/10.1016/0304-3975(82)90125-6
21. König, D.: Über eine Schlussweise aus dem Endlichen ins Unendliche. Acta Sci. Math. (Szeged) **3**(2–3), 121–130 (1927)
22. Laarman, A., Pater, E., van de Pol, J., Hansen, H.: Guard-based partial-order reduction. Int. J. Softw. Tools Technol. Transf. **18**(4), 427–448 (2016). https://doi.org/10.1007/s10009-014-0363-9
23. Lann, G.L.: Distributed systems—towards a formal approach. In: IFIP, 1977, pp. 155–160 (1977)
24. 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
25. Milner, R.: A Calculus of Communicating Systems, LNCS, vol. 92. Springer, Berlin (1980)
26. Neele, T.: Reductions for Parity Games and Model Checking. Ph.D. thesis, Eindhoven University of Technology (2020)
27. Neele, T., Valmari, A., Willemse, T.A.C.: The inconsistent labelling problem of stutter-preserving partial-order reduction. In: FoSSaCS 2020, LNCS, vol. 12077, pp. 482–501 (2020). https://doi.org/10.1007/978-3-030-45231-5_25
28. Neele, T., Valmari, A., Willemse, T.A.C.: A detailed account of the inconsistent labelling problem of stutter-preserving partial-order reduction. Log. Methods Comput. Sci. **17**, 1–27 (2021). https://doi.org/10.46298/lmcs-17(3:8)2021
29. Neele, T., Willemse, T.A.C., Groote, J.F.: Finding compact proofs for infinite-data parameterised Boolean equation systems. Sci. Comput. Program. **188**, 102389 (2020). https://doi.org/10.1016/j.scico.2019.102389
30. Neele, T., Willemse, T.A.C., Wesselink, W.: Partial-order reduction for parity games with an application on parameterised Boolean equation systems. In: TACAS 2020, LNCS, vol. 12079, pp. 307–324 (2020). https://doi.org/10.1007/978-3-030-45237-7_19
31. Pelánek, R.: BEEM: benchmarks for explicit model checkers. In: SPIN 2007, LNCS, vol. 4595, pp. 263–267 (2007). https://doi.org/10.1007/978-3-540-73370-6_17
32. 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
33. Peled, D.: Combining partial order reductions with on-the-fly model-checking. Formal Methods Syst. Des. **8**(1), 39–64 (1996). https://doi.org/10.1007/BF00121262
34. Ploeger, B., Wesselink, J.W., Willemse, T.A.C.: Verification of reactive systems via instantiation of parameterised Boolean equation systems. Inf. Comput. **209**(4), 637–663 (2011). https://doi.org/10.1016/j.ic.2010.11.025
35. 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
36. Siegel, S.F.: What's wrong with on-the-fly partial order reduction. In: CAV 2019, LNCS, vol. 11562, pp. 478–495 (2019). https://doi.org/10.1007/978-3-030-25543-5_27
37. Valmari, A.: Error detection by reduced reachability graph generation. In: APN 1988, pp. 95–112 (1988)
38. Valmari, A.: A stubborn attack on state explosion. Formal Methods Syst. Des. **1**(4), 297–322 (1992). https://doi.org/10.1007/BF00709154
39. 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
40. Valmari, A., Hansen, H.: Stubborn set intuition explained. ToPNoC **10470**(12), 140–165 (2017). https://doi.org/10.1007/978-3-662-55862-1_7
41. Valmari, A., Vogler, W.: Fair testing and stubborn sets. Int. J. Softw. Tools Technol. Transf. **20**(5), 589–610 (2018). https://doi.org/10.1007/s10009-017-0481-2
42. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1–2), 135–183 (1998). https://doi.org/10.1016/S0304-3975(98)00009-7