A cure for stuttering parity games

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

A cure for stuttering parity games

Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse

12/05

ISSN 0926-4515

editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 12-05
Eindhoven, March 2012

A cure for stuttering parity games

Sjoerd Cranen, Jeroen J.A. Keiren, and Tim A.C. Willemse

Department of Computer Science and Mathematics
Eindhoven University of Technology
PO Box 513, 5600MB Eindhoven, The Netherlands

Abstract. We define governed stuttering bisimulation for parity games, weakening stuttering bisimulation by taking the ownership of vertices into account only when this might lead to observably different games. We show that governed stuttering bisimilarity is an equivalence for parity games and allows for a natural quotienting operation. Moreover, we prove that all pairs of vertices related by governed stuttering bisimilarity are won by the same player in the parity game. Thus, our equivalence can be used as a preprocessing step when solving parity games. Governed stuttering bisimilarity can be decided in $\mathcal{O}(n^2m)$ time for parity games with n vertices and m edges. Our experiments indicate that governed stuttering bisimilarity is mostly competitive with stuttering equivalence on parity games encoding typical verification problems.

1 Introduction

Parity games [10, 21, 25] are played by two players (represented by $\diamond$ and $\Box$) on a directed graph in which every vertex is owned by one of the players, and vertices are assigned a priority. The game is played by moving a single token along the edges in the graph; the choice where to move next is dictated by the player owning the vertex on which the token currently resides. Both players try to play such that the thus created infinite path is *winning* for them, and a vertex is won by the player that can play such that, however the opponent plays, every path from that vertex is won by her. The winner of a vertex is uniquely determined [10, 21, 25] and partitioning the graph in vertices that are won by player $\diamond$ and those won by player $\Box$ is referred to as *solving* the parity game.

The parity game framework is a key instrument in solving practical verification and synthesis problems, see [10, 2]. Its practical significance is mirrored by its role in searching for the true complexity of model checking: modal μ-calculus model checking is polynomially reducible to parity game solving, and *vice versa*. Despite the apparent simplicity of the latter problem, the precise complexity of solving parity games is still open: the problem is known to be in NP $\cap$ coNP, and more specifically in UP $\cap$ coUP [16], suggesting there just might exist a polynomial time algorithm. Indeed, non-trivial classes of parity games have been identified that admit polynomial time solving algorithms, see *e.g.* [4, 22].

In the past decade, several advanced algorithms for solving parity games have been designed. These include algorithms exponential in the number of priorites, such as Jurdziński's *small progress measures* algorithm [17] and Schewe's *bigstep* algorithm [23], as well as the sub-exponential algorithm due to Jurdziński *et al.* [18]. Orthogonally to

the algorithmic improvements, heuristics have been devised that may speed up solving parity games that occur in practice, see [11] for an overview. Such heuristics work particularly well for verification problems, which give rise to games with only few different priorities.

The heuristic that we consider in this paper, following, *e.g.*, Fritz and Wilke's study of *delayed simulation* [13], is based on the use of fine-grained equivalence relations that approximate the solution to a parity game. The idea is to recast the solving problem as the problem of deciding *winner equivalence* between vertices: two vertices in a parity game are equivalent whenever they are won by the same player. Finding equivalence relations that refine winner equivalence and that are decidable in polynomial time yields a preprocessing step that can be used to reduce games prior to solving.

From a practical viewpoint, we are particularly interested in those simulation and equivalence relations that strike a favourable balance between their power to compress the game graph and their computational complexity. Stuttering bisimulation [7] for Kripke Structures is among a select number of candidates worth considering, with an $\mathcal{O}(nm)$ time complexity (n being the number of vertices and m the number of edges). Indeed, as earlier experiments [9] indicate, off-the-shelf stuttering bisimulation reduction algorithms can be competitive when compared to modern available parity game solvers. Stuttering bisimulation, however, is inept when faced with alternations between players along the possible plays: it cannot relate vertices belonging to different players. Turn-based games, controller synthesis problems *e.g.* [2], and, in general, constructs such as $\Box\Diamond\phi$ and $\Diamond\Box\phi$ in μ-calculus verification, all give rise to such parity games.

A natural question is, therefore, whether stuttering bisimulation can at all be modified so that it is able to relate vertices that belong to different players. We positively answer this question in this paper by defining a relation, which we dub *governed stuttering bisimulation* (reflecting that a player's ruling capabilities are taken as primitive), which we show to be strictly weaker than stuttering bisimilarity. In addition, we prove that governed stuttering bisimilarity:

- is an equivalence relation on parity games.
- refines winner equivalence.
- is decidable in $\mathcal{O}(n^2m)$ time using a partition refinement algorithm.

The time complexity for deciding governed stuttering bisimilarity is a factor n worse than that for stuttering bisimilarity; this is due to a single type of class in a partition for which our algorithm requires $\mathcal{O}(mn)$ rather than $\mathcal{O}(m)$ time to check its stability. Our experiments, however, indicate that this factor does not manifest itself in practice; in fact, our algorithm is mostly competitive with the one for stuttering bisimilarity.

Structure of the paper. Section 2 briefly introduces the parity game framework. We recall the definition of stuttering bisimulation and we define governed stuttering bisimulation in Section 4. In Section 5, we show that governed stuttering bisimulation is an equivalence relation, we show it refines winner equivalence, and we address its decidability. We discuss our experiments with a prototype implementation of our algorithm for deciding governed stuttering bisimulation in Section 6. Related work is discussed in Section 7, and future work is described in Section 8.

2 Preliminaries

A parity game is a two-player graph game, played by two players on a directed graph. The game is formally defined as follows.

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

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

If i is a player, then $\neg i$ denotes the opponent of i, *i.e.*, $\neg\diamond = \Box$ and $\neg\Box = \diamond$. A sequence of vertices $v_1, \ldots, v_n$ for which $v_m \rightarrow v_{m+1}$ for all $1 \leq m < n$ is a *path*, and may be denoted using angular brackets: $\langle v_1 \ldots v_n \rangle$. The concatenation $p \cdot q$ of paths p and q is again a path. Infinite paths are defined in a similar manner. We use p_n to denote the n^{th} vertex in a path p.

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

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

A path p of length n is *consistent* with a strategy $\sigma \in \mathbb{S}^*_i$, denoted $\sigma \Vdash p$, if and only if for all $1 \leq j < n$ it is the case that if σ is defined for $\langle p_1 \ldots p_j \rangle$, then $p_{j+1} = \sigma(\langle p_1 \ldots p_j \rangle)$. The definition of consistency is extended to infinite paths in the obvious manner. A strategy $\sigma \in \mathbb{S}^*_i$ is said to be a *winning strategy* from a vertex v if and only if i is the winner of every path consistent with σ. A vertex is won by i if i has a winning strategy from that vertex. Parity games are memoryless determined [10], *i.e.* each vertex in the game is won by exactly one player, and it suffices to play a memoryless strategy.

In this paper, we are concerned with relations partitioning the vertices in a parity game such that all related vertices are won by the same player. Let R be a relation over a set V. For $v, w \in V$ we write $v \ R \ w$ for $(v, w) \in R$. For an equivalence relation R, and vertex $v \in V$ we define $[v]_R$, the equivalence class of v under R, as $\{v' \in V \mid v \ R \ v'\}$. The set of equivalence classes of V under R is denoted $V_{/R}$. A collection $\{B_i \mid i \in I\}$, with $\emptyset \neq B_i \subseteq V$, is called a *partition* of V if $\bigcup_{i \in I} B_i = V$ and for $i \neq j : B_i \cap B_j = \emptyset$. An element B_i of a partition is called a *block*. If P and Q are partitions of V then Q *refines* P if $\forall B_i \in Q : \exists B_j \in P : B_i \subseteq B_j$. We

use the notions of equivalence relation and partition interchangeably, and occasionally write $v\ P\ v'$ rather than $v, v' \in B$ for some $B \in P$.

Determinacy of parity games effectively induces a partition on the set of vertices V in those vertices won by player $\diamond$ and those vertices won by player $\Box$. This partition is the natural equivalence relation on V.

Definition 2 (Winner equivalence). *Let $(V, \to, \Omega, \mathcal{P})$ be a parity game. Vertices $v, w \in V$ are* winner equivalent*, denoted $v \sim w$ iff v and w are won by the same player.*

3 Properties of parity games

We first formalise some intuitions about parity games. In the following lemmata, let $v \in V$, i a player, $U, U' \subseteq V$, R an equivalence relation and $\mathcal{U} \subseteq V_{/R}$.

One of the most basic properties we expect to hold is that a player can force the play towards some given set of vertices, or otherwise her opponent can force the play to the complement of that set.

Lemma 1. $v\ {}_i\!\!\mapsto_R U \lor v\ {}_{\neg i}\!\!\mapsto_R V \setminus U$.

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

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

In a similar train of thought, we expect that if from a single vertex, each player can force play towards some target set, then the players' target sets must contain related vertices.

Lemma 2. $v\ {}_i\!\!\mapsto_R U \land v\ {}_{\neg i}\!\!\mapsto_R U' \implies \exists u \in U, u' \in U' : u\ R\ u' \lor u\ R\ v \lor u'\ R\ v$.

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

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

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

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

Notice that this lemma relies on R being an equivalence relation. In particular, transitivity is used in the repetition of the argument. If we consider sets of *classes* in R that a player can force to, rather than arbitrary sets of vertices, we arrive at a much stronger version of Lemma 1; the result follows directly from lemmata 1 and 2.

Corollary 1. $v \,{}_i\!\not\mapsto_R \mathcal{U} \implies v \,{}_{\neg i}\!\mapsto_R V_{/R} \setminus \mathcal{U}$.

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

Lemma 3. $v \,{}_i\!\mapsto_R \iff v \,{}_{\neg i}\!\not\mapsto_R V \setminus [v]_R$

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

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

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

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

Lemma 4. $(v \,{}_i\!\mapsto_R U \wedge (\forall u \in U \setminus U' : v \; R \; u \wedge u \,{}_i\!\mapsto_R U')) \implies v \,{}_i\!\mapsto_R U'$

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

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

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

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

4 Governed stuttering bisimulation

In [9] we introduced *stuttering bisimulation* for parity games. Informally, stuttering bisimulation compresses subsequences of "identical" vertices along a path p in a parity game, such that the path retains the essentials of the graph's branching structure. Identical vertices are basically vertices with the same priority, owned by the same player.

Before we give the formal definition of stuttering bisimulation, we first introduce some notation. Let $(V, \to, \Omega, \mathcal{P})$ be a parity game. From hereon, let $U \subseteq V$ be arbitrary sets of vertices; we write $v \to U$ if there exists a $u \in U$ such that $v \to u$.

Let $R \subseteq V \times V$ be a relation on the set of vertices. The generalised transition relation $v \mapsto_R U$, defined below, formalises that U is eventually reached from v by some computation path through R-related nodes. Dually, $v \mapsto_R$ expresses that v is the start of an infinite computation path along vertices related through R.

$$\begin{aligned} v \mapsto_R U &\stackrel{\mu}{=} \exists u : v \to u \wedge (u \in U \vee (v \; R \; u \wedge u \mapsto_R U)) \\ v \mapsto_R \quad &\stackrel{\nu}{=} \exists u : v \to u \wedge v \; R \; u \wedge u \mapsto_R \end{aligned}$$

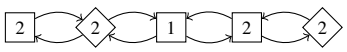
We next formalise the notion of stuttering bisimulation, deviating notationally from [9]; the definitions, however, are easily seen to coincide and the modifications are standard. Our main reason for deviating from [9] is that the presented definition facilitates explaining the intuition of its generalisation to governed stuttering bisimulation.

Definition 3 (Stuttering bisimulation [9]). *Let $(V, \to, \Omega, \mathcal{P})$ be a parity game. Let $R \subseteq V \times V$ be an equivalence relation on vertices; R is a* stuttering bisimulation *if $v \; R \; v'$ implies*

a) $\Omega(v) = \Omega(v')$ and $\mathcal{P}(v) = \mathcal{P}(v')$;
b) $v \to \mathcal{C}$ implies $v' \mapsto_R \mathcal{C}$, for all $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$.
c) $v \mapsto_R$ iff $v' \mapsto_R$;

Two states v and v' are said to be stuttering bisimilar*, denoted $v \simeq v'$ iff there is a stuttering bisimulation relation R, such that $v \; R \; v'$.*

Our objective is to weaken stuttering bisimulation so that it will be able to relate vertices of different players. However, we cannot simply weaken clause *a)* to $\Omega(v) = \Omega(v')$ without modifying the remaining clauses, as this would enable us to relate vertices won by different players, as the below parity game demonstrates:



The suggested weakening of clause *a)* would allow us to relate all vertices with priority 2; the two left vertices, however are won by player $\diamond$, whereas the other vertices are won by player $\Box$.

The problem in the above example is that the computation paths that appear in clauses *b)* and *c)* may consist of vertices owned by different players. This means that a fixed player is at the mercy of her opponent to stay on a computation path: the opponent may simply choose an alternative next vertex if that would better suit her. We are therefore forced to reason about computation trees, taking all the opponent's choices into

account. Effectively, clause *b)* must be strengthened to ensure that a player eventually reaches class $\mathcal{C}$ along some computation tree, and clause *c)* must be strengthened to ensure that a player can construct an infinite computation tree not leaving its own class.

We first extend our notation to facilitate reasoning about computation trees rather than computation paths. Given a memoryless strategy σ for some player, the ability to move from vertex v to another vertex u depends on this strategy.

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

From the viewpoint of a fixed player and her memoryless strategy σ, a token may be moved along the edges of a computation tree that only branches at vertices owned by her opponent. The notation $v \mathrel{{}_\sigma\!\!\mapsto} U$, defined below, formalises that all plays according to σ eventually reach the set of vertices U. This notation is generalised to $v \mathrel{{}_\sigma\!\!\mapsto_R} U$, enabling us to express that, additionally, all plays allowed by σ reach U immediately when they follow an edge to a vertex that is no longer related under relation R. The notation $v \mathrel{{}_\sigma\!\!\mapsto_R}$ is basically its dual; it expresses that all plays allowed by σ can reach only vertices related under R to the previous vertex in that play:

$$\begin{aligned} v \mathrel{{}_\sigma\!\!\mapsto_R} U &\stackrel{\mu}{=} \forall u : v \mathrel{{}_\sigma\!\!\rightarrow} u \implies u \in U \vee (v \mathrel{R} u \wedge u \mathrel{{}_\sigma\!\!\mapsto_R} U) \\ v \mathrel{{}_\sigma\!\!\mapsto_R} &\stackrel{\nu}{=} \forall u : v \mathrel{{}_\sigma\!\!\rightarrow} u \implies v \mathrel{R} u \wedge u \mathrel{{}_\sigma\!\!\mapsto_R} \end{aligned}$$

If the strategy is unimportant to the purpose at hand, we abstract from the specific strategy that is used and reason only in terms of a player i having a strategy with the capability of forcing a play to a set of vertices U, and, dually, for i to be able to force the play to *diverge* within a class of R:

$$\begin{aligned} x \mathrel{{}_i\!\!\mapsto_R} U &= \exists \sigma \in \mathbb{S}_i : x \mathrel{{}_\sigma\!\!\mapsto_R} U \\ x \mathrel{{}_i\!\!\mapsto_R} &= \exists \sigma \in \mathbb{S}_i : x \mathrel{{}_\sigma\!\!\mapsto_R} \end{aligned}$$

We omit R if it is the relation that relates all vertices in V. Note that $v \mathrel{{}_i\!\!\mapsto_R} \emptyset$ never holds. On the other hand, $v \mathrel{{}_i\!\!\mapsto_R} V$ and $v \mathrel{{}_i\!\!\mapsto}$ are trivially true. We write $v \mathrel{{}_i\!\!\not\mapsto_R} U$ for $\neg(v \mathrel{{}_i\!\!\mapsto_R} U)$; likewise for all other arrows. If $\mathcal{U} \subseteq V_{/R}$, then we write $v \mathrel{{}_i\!\!\mapsto_R} \mathcal{U}$ to denote $v \mathrel{{}_i\!\!\mapsto_R} \bigcup_{\mathcal{C} \in \mathcal{U}} \mathcal{C}$.

Definition 4 (Governed stuttering bisimulation). *Let $(V, \rightarrow, \Omega, \mathcal{P})$ be a parity game. Let $R \subseteq V \times V$ be an equivalence relation. Then R is a* governed stuttering bisimulation *if $v \mathrel{R} v'$ implies*

a) $\Omega(v) = \Omega(v')$;
b) $v \rightarrow \mathcal{C}$ implies $v' \mathrel{{}_{\mathcal{P}(v)}\!\!\mapsto_R} \mathcal{C}$, for all $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$.
c) $v \mathrel{{}_i\!\!\mapsto_R}$ iff $v' \mathrel{{}_i\!\!\mapsto_R}$ for $i \in \{\diamond, \Box\}$.

Vertices v and v' are governed stuttering bisimilar*, denoted $v \simeq v'$, iff a governed stuttering bisimulation R exists such that $v \mathrel{R} v'$.*

If we additionally require that $\mathcal{P}(v) = \mathcal{P}(v')$, we find that $v \mapsto_R U$ iff $v \mathrel{{}_{\mathcal{P}(v)}\!\!\mapsto_R} U$, and, likewise, $v \mapsto_R$ iff $v \mathrel{{}_{\mathcal{P}(v)}\!\!\mapsto_R}$. This is the basis for the following proposition.

Proposition 1. *Let $R \subseteq V \times V$ be a governed stuttering bisimulation, such that $v\ R\ v'$ implies $\mathcal{P}(v) = \mathcal{P}(v')$. Then R is a stuttering bisimulation.*

Example 1. Consider the parity game depicted in Figure 1a. The equivalence relation that relates vertices with equal priorities is a governed stuttering bisimulation. Stuttering bisimulation does not relate any of the vertices.

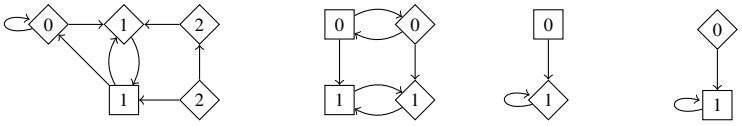


Fig. 1: All vertices in (a) with the same priorities can be related using governed stuttering bisimilarity. Both (c) and (d) are minimal representations of (b).

5 Properties of governed stuttering bisimulation

We next study three key properties of governed stuttering bisimulation, *viz.*, governed stuttering bisimilarity is an equivalence on parity games, it refines winner equivalence and it is decidable in polynomial time.

5.1 Governed stuttering bisimilarity is an equivalence

Proving that $\simeq$ is an equivalence relation on parity games is far from straightforward: transitivity no longer bows to the standard proof strategies that work for stuttering bisimilarity and branching bisimilarity [24]. As a result of the asymmetry in the use of two different transition relations in clause *b)* of Definition 4, proving that the equivalence closure of the union of two governed stuttering bisimulation relations is again a governed stuttering bisimulation relation is equally problematic.

The strategy we pursue is as follows. We characterise governed stuttering bisimulation, in two steps, by a set of symmetric requirements. The obtained alternative characterisation is then used in our equivalence proof. These alternative characterisations do not facilitate the reuse of standard proof strategies, but they are instrumental in the technically involved proof that the equivalence closure of two governed stuttering bisimulation relations is again a governed stuttering bisimulation relation. Apart from being convenient technically, the characterisations offer more insight into the nature of governed stuttering bisimilarity. Hence, instead of providing the details of our equivalence proof, we focus on the alternative characterisations of governed stuttering bisimulation.

Our result below states that we can rephrase condition *b)* of governed stuttering bisimulation by requiring that a fixed player must have the same power to force the play from any pair of related vertices to reach an arbitrary class. Thus, we abstract from the player that takes the initiative to leave its class in one step.

Theorem 1. *Let $R \subseteq V \times V$ and $v, v' \in V$. Then R is a governed stuttering bisimulation iff R is an equivalence relation and $v\ R\ v'$ implies:*

a) $\Omega(v) = \Omega(v')$*;*
b) $v \ {}_i\!\mapsto_R \mathcal{C}$ *iff* $v' \ {}_i\!\mapsto_R \mathcal{C}$ *for all* $i \in \{\Diamond, \Box\}, \mathcal{C} \in V_{/R} \setminus \{[v]_R\}$*;*
c) $v \ {}_i\!\mapsto_R$ *iff* $v' \ {}_i\!\mapsto_R$ *for all* $i \in \{\Diamond, \Box\}$.

Proof. The proof for the implication from right to left is immediate. For the other direction, assume that R is a governed stuttering bisimulation. Observe that it suffices to prove clause *b*; the other two are in full agreement with Definition 4.

Let i be an arbitrary player, and suppose that $v \ {}_i\!\mapsto_R \mathcal{C}$ for some $v \in V$ and $\mathcal{C} \in V_{/R} \setminus \{[v]_R\}$. Obviously, there must be some $u \in [v]_R$ such that $u \to \mathcal{C}$. Let u be such, and distinguish the following two cases:

- Case $\mathcal{P}(u) = i$. It follows directly from Definition 4 that $v' \ {}_i\!\mapsto_R \mathcal{C}$.
- Case $\mathcal{P}(u) \neq i$. From Lemma 3 it follows that $v \ {}_{\neg i}\!\not\mapsto_R$ as $v \ {}_i\!\mapsto_R \mathcal{C}$, and that $v \ {}_i\!\not\mapsto_R$ because $u \ {}_{\neg i}\!\mapsto_R \mathcal{C}$. By Definition 4, it then also holds that $v' \ {}_i\!\not\mapsto_R$ and $v' \ {}_{\neg i}\!\not\mapsto_R$. Towards a contradiction, suppose that $v' \ {}_i\!\not\mapsto_R \mathcal{C}$. By Lemma 1 it must then be the case that $v' \ {}_{\neg i}\!\mapsto_R V \setminus \mathcal{C}$. Because of this, and $v \ {}_{\neg i}\!\not\mapsto_R$ and $v \ {}_i\!\not\mapsto_R$ it follows that there must be some u' such that $u' \to V \setminus \mathcal{C} \setminus [v]_R$. We again distinguish two cases:
  - Case $\mathcal{P}(u') = i$. Then $u' \ {}_i\!\to V \setminus \mathcal{C} \setminus [v]_R$, and by definition of governed stuttering bisimulation $u \ {}_i\!\mapsto_R V \setminus \mathcal{C} \setminus [v]_R$, which contradicts $u \ {}_{\neg i}\!\mapsto_R \mathcal{C}$ according to Lemma 3.
  - Case $\mathcal{P}(u') \neq i$. Then $u' \ {}_{\neg i}\!\to V \setminus \mathcal{C} \setminus [v]_R$, and by definition of governed stuttering bisimulation $v \ {}_{\neg i}\!\mapsto_R V \setminus \mathcal{C} \setminus [v]_R$, which is a contradiction to $v \ {}_i\!\mapsto_R \mathcal{C}$ according to Lemma 3. □

While the above alternative characterisation of governed stuttering bisimulation is now fully symmetric, the restriction on the class $\mathcal{C}$ that is considered in clause *b)* turns out to be too strong to facilitate our proof that $\simeq$ is an equivalence relation. We therefore generalise this clause once more to reason about sets of classes. A perhaps surprising side-result of this generalisation is that the divergence requirement of clause *c)* becomes superfluous. Note that this last generalisation is not trivial, as $v \ {}_i\!\mapsto_R \{\mathcal{C}_1, \mathcal{C}_2\}$ is in general neither equivalent to saying that $v \ {}_i\!\mapsto_R \mathcal{C}_1$ and $v \ {}_i\!\mapsto_R \mathcal{C}_2$, nor $v \ {}_i\!\mapsto_R \mathcal{C}_1$ or $v \ {}_i\!\mapsto_R \mathcal{C}_2$.

Theorem 2. *Let $R \subseteq V \times V$ and $v, v' \in V$. Then R is a governed stuttering bisimulation iff R is an equivalence relation and $v\ R\ v'$ implies:*

a) $\Omega(v) = \Omega(v')$*;*
b) $v \ {}_i\!\mapsto_R \mathcal{U}$ *iff* $v' \ {}_i\!\mapsto_R \mathcal{U}$ *for all* $i \in \{\Diamond, \Box\}, \mathcal{U} \subseteq V_{/R} \setminus \{[v]_R\}$.

Proof. We show that the above clause *b* is equivalent to clauses *b* and *c* from Theorem 1. We split the proof into an *if*-part and an *only-if*-part.

$\Leftarrow$ Clause *b* follows immediately (if $v \ {}_i\!\mapsto_R \mathcal{C}$, take $\mathcal{U} = \{\mathcal{C}\}$). Clause *c* follows directly from Theorem 1 if we substitute $V_{/R} \setminus \{[v]_R\}$ for $\mathcal{U}$.

$\Rightarrow$ Let R be a governed stuttering bisimulation relation and let $v, v' \in V$ such that $v\ R\ v'$. Assume that $v \ {}_i\!\mapsto_R \mathcal{U}$ for some $\mathcal{U} \subseteq V_{/R} \setminus \{[v]_R\}$. Let $S = \{u \in [v]_R \mid u \ {}_i\!\to \mathcal{U}\}$. We distinguish the following two cases:

$\exists u \in S : \mathcal{P}(u) = i$. Let u be such. There is a class $\mathcal{C} \in \mathcal{U}$ such that $u \,{}_i\!\!\rightarrow \mathcal{C}$ (in particular, $u \,{}_i\!\!\mapsto_R \mathcal{C}$). By Theorem 1 then also $v' \,{}_i\!\!\mapsto_R \mathcal{C}$, from which $v' \,{}_i\!\!\mapsto_R \mathcal{U}$ follows immediately.

$\forall u \in S : \mathcal{P}(u) \neq i$. From $v \,{}_i\!\!\mapsto_R \mathcal{U}$ we derive, using Lemma 3, that $v \,{}_{\neg i}\!\!\not\rightarrow_R$. By Theorem 1 it follows that $v' \,{}_{\neg i}\!\!\not\rightarrow_R$. We also know that $v' \,{}_i\!\!\not\rightarrow_R$, because otherwise for all nodes u in S we have $u \,{}_i\!\!\mapsto_R$ by Theorem 1, which cannot be the case because $\mathcal{P}(u) \neq i$.

Towards a contradiction, suppose that $v' \,{}_i\!\!\not\rightarrow_R \mathcal{U}$, then by Lemma 1 we have $v' \,{}_{\neg i}\!\!\mapsto_R V_{/R} \setminus \mathcal{U}$. Because $v' \,{}_{\neg i}\!\!\not\rightarrow_R$ and $v' \,{}_i\!\!\not\rightarrow_R$, we find $v' \,{}_{\neg i}\!\!\mapsto_R V_{/R} \setminus \mathcal{U} \setminus \{[v]_R\}$ and some $u' \in S$ such that $u' \,{}_{\neg i}\!\!\rightarrow V_{/R} \setminus \mathcal{U} \setminus \{[v]_R\}$ to witness it. In particular, $u' \,{}_{\neg i}\!\!\rightarrow \mathcal{C}$ for some $\mathcal{C} \in V_{/R} \setminus \mathcal{U} \setminus \{[v]_R\}$, and by Theorem 1 also $v \,{}_{\neg i}\!\!\mapsto_R \mathcal{C}$. This contradicts $v \,{}_i\!\!\mapsto_R \mathcal{U}$ by Lemma 2. □

Note that the divergence requirement $v \,{}_i\!\!\mapsto_R$ *iff* $v' \,{}_i\!\!\mapsto_R$ can be recovered by instantiating set $\mathcal{U}$ by $V_{/R} \setminus \{[v]_R\}$ for player $\neg i$ in the above theorem. The last characterisation enables us to prove that that $\simeq$ is an equivalence relation. As the proof is non-standard we introduce several lemmata leading up to this result.

We define the composition $(R \circ S)$ of two relations R and S such that $v \; (R \circ S) \; w$ iff there exists some u such that $v \; S \; u$ and $u \; R \; w$. Vertices v and w are related under union, *i.e.* $v \; (R \cup S) \; w$ iff $v \; R \; w$ or $v \; S \; w$.

Lemma 5. *If R and S satisfy the first clause from Theorem 2, then so does $(R \cup S)^*$.*

Proof. If $v, v' \in V$ are related under $(R \cup S)^*$, then there exists a sequence of vertices $v, u_0, u_1, \ldots, u_n, v'$ such that each consecutive pair in the sequence is related under R or S. By transitivity of $=$ we then have $\Omega(v) = \Omega(v')$. □

We introduce the following notion to allow for a structured proof of the second property of governed stuttering bisimulation.

Definition 5. *Let R be a relation. We call R weakly transferring if it is reflexive and $v' \; R \; v$ implies for all $v, v' \in V$ and $U \subseteq V$ that*

$$v \,{}_i\!\!\mapsto_{R^*} U \implies \exists U' \subseteq V : v' \,{}_i\!\!\mapsto_{R^*} U' \wedge \forall u' \in U' : \exists u \in U : (u \; R \; u' \wedge u' \; R \; u).$$

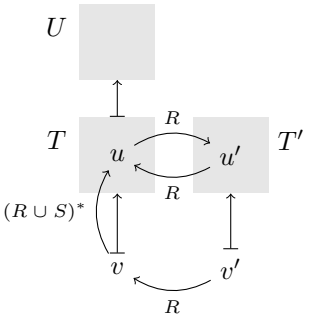
Lemma 6. *Let R and S be weakly transferring relations, and $v, v' \in V$ s.t. $v' \; R \; v$, then*

$$v \,{}_i\!\!\mapsto_{(R \cup S)^*} U \implies \exists U' \subseteq V : \; v' \,{}_i\!\!\mapsto_{(R \cup S)^*} U' \wedge \forall u' \in U' : \exists u \in U : (u \; R \; u' \wedge u' \; R \; u). \quad (1)$$

Proof. If $v \,{}_i\!\!\mapsto_{R^*} U$, then this follows immediately from $v' \; R \; v$ and the fact that R itself is weakly transferring.

Suppose now that $v \,{}_i\!\!\not\rightarrow_{R^*} U$, and let $\sigma \in \mathbb{S}_i^*$ be such that $v \,{}_\sigma\!\!\mapsto_{(R \cup S)^*} U$. It must be the case that $v \,{}_\sigma\!\!\mapsto_{R^*} T$ for some smallest $T \subseteq V \setminus [v]_R$, and furthermore T must be

such that $\forall u \in T : u \ {}_{\sigma}\!\mapsto_{(R \cup S)^*} U \ \vee u \in U$. Because R is weakly transferring, we must have $v' \ {}_{i}\!\mapsto_{R^*} T'$ for some T' such that $\forall u' \in T' : \exists u \in T : (u \ R \ u' \ \wedge \ u' \ R \ u)$. Furthermore, because $v' \ R \ v$, and $v \ (R \cup S)^* \ u$ for every $u \in T \setminus U$, we also have $v' \ (R \cup S)^* \ u'$ for all $u' \in T' \setminus U$.



We can repeat the argument for every $u' \in T'$ to find $u' \ {}_{i}\!\mapsto_{(R \cup S)^*} U'$ for some $U' \subseteq V$ for which $\forall u' \in U' : \exists u \in U : u \ R \ u'$; we either end up at the trivial case above, or repeat the argument again. The argument cannot be repeated infinitely often, because then false would be the least solution of $v \ {}_{i}\!\mapsto_{(R \cup S)^*} U$. Using Lemma 4 we now obtain $v' \ {}_{i}\!\mapsto_{(R \cup S)^*} U'$, concluding our proof.

□

Lemma 7. *If R and S are weakly transferring relations, then $(R \cup S)$ is too, and so is $(R \circ S)$.*

Proof. Let R and S be two weakly transferring relations. We give two separate proofs for $(R \cup S)$ and $(R \circ S)$ being weakly transferring.

Notice that $(R \cup S)^* = (R \circ S)^*$ because R and S are reflexive.

- $R \cup S$ *is weakly transferring*. Suppose that $v' \ (R \cup S) \ v$. In that case $v' \ R \ v$ or $v' \ S \ v$. Suppose $v' \ R \ v$ (the other case is completely symmetrical), then Lemma 6 immediately gives us the desired result because $R \subseteq (R \cup S)$.
- $R \circ S$ *is weakly transferring*. If $v' \ (R \circ S) \ v$, then there must be some $x \in V$ such that $v' \ S \ x$ and $x \ R \ v$. Suppose that $v \ {}_{i}\!\mapsto_{(R \cup S)^*} U$ for some $U \subseteq V$. We use Lemma 6 to obtain some $U' \subseteq V$ such that

$$x \ {}_{i}\!\mapsto_{(R \cup S)^*} U' \wedge \forall u' \in U' : \exists u \in U : (u \ R \ u' \wedge u' \ R \ u).$$

We can prove something similar to Lemma 6 for S to obtain a set $U'' \subseteq V$ such that

$$v' \ {}_{i}\!\mapsto_{(R \cup S)^*} U'' \wedge \forall u'' \in U'' : \exists u' \in U' : (u' \ S \ u'' \wedge u'' \ S \ u').$$

Combining the two, we find

$$v' \ {}_{i}\!\mapsto_{(R \cup S)^*} U'' \wedge \forall u'' \in U'' : \exists u \in U : (u \ (R \circ S) \ u'' \wedge u'' \ (R \circ S) \ u).$$

Because $(R \cup S)^* = (R \circ S)^*$, we have hereby proven that $(R \circ S)$ is weakly transferring. □

Lemma 8. *If R and S are equivalence relations satisfying the second clause from Theorem 2, then $(R \cup S)^*$ also satisfies that clause.*

Proof. Notice that an equivalence relation R is weakly transferring if and only if the second clause of Theorem 2 holds for R. Using Lemma 7 it is easy to see that $(R \cup S)^*$ is also weakly transferring. Furthermore, $(R \cup S)^*$ is an equivalence relation because R and S are, and therefore the second clause of Theorem 2 must also hold for $(R \cup S)^*$. □

Lemma 9. *If R and S are governed stuttering bisimulation relations, then $(R \cup S)^*$ satisfies the third clause of Theorem 2.*

Proof. Assume $v \ R \ v'$, and $v \, {}_i\!\mapsto_{(R \cup S)^*}$. Let $\sigma \in \mathbb{S}_i^*$ be such that $v \, {}_\sigma\!\mapsto_{(R \cup S)^*}$, and let $U \subseteq V$ be the smallest set such that $v \, {}_\sigma\!\mapsto_R U$. Then it follows directly from the definition of governed stuttering bisimulation that $v' \, {}_i\!\mapsto_R U'$ for some $U' \subseteq V$ such that $\forall u' \in U' : \exists u \in U : u \ R \ u'$. It must be the case that $u \ (R \cup S)^* \ v$ for all $u \in U$, so by transitivity of R and $(R \cup S)^*$, $v' \ (R \cup S)^* \ u'$ for all $u' \in U'$. This argument can be repeated ad infinitum.

Using a symmetric proof, we can show that if $v \ S \ v'$ and $v \, {}_i\!\mapsto_{(R \cup S)^*}$ imply that $v' \, {}_i\!\mapsto_{(R \cup S)^*}$. If $v \ (R \cup S)^* \ v'$, then there must be a sequence $v, u_0, \ldots u_n, v'$ in which every consecutive pair of vertices is related under R or under S. The desired result then follows from transitivity of implication. □

Lemma 10. *If R and S are governed stuttering bisimulation relations, then so is the relation $(R \cup S)^*$.*

Proof. Obviously, $(R \cup S)^*$ is an equivalence relation, because R and S are too. Lemmas 5, 8 and 9 show that $(R \cup S)^*$ satisfies the remaining criteria for governed stuttering bisimulation. □

Finally, we prove that $\simeq$ is an equivalence relation.

Theorem 3. *$\simeq$ is an equivalence relation.*

Proof. Because the identity relation is a governed stuttering bisimulation relation, $\simeq$ is reflexive. It is symmetric, because if $v \simeq v'$ then there is a governed stuttering bisimulation relation R such that $v \ R \ v'$, but then also $v' \ R \ v$, hence $v' \simeq v$. It is transitive, because if $v \simeq v'$ and $v' \simeq v''$, then there are governed stuttering bisimulation relations R and S such that $v \ R \ v'$ and $v' \ S \ v''$. Obviously $v \ (R \cup S)^* \ v''$, and because $(R \cup S)^*$ is a governed stuttering bisimulation relation according to Lemma 10, also $v \simeq v''$. □

5.2 Quotienting

The main reason for studying equivalence relations for parity games is that they may offer the prospect of minimising the parity game by collapsing vertices that are considered equivalent. The resulting minimised structure is referred to as the quotient. However, not all equivalence relations admit such a quotienting operation; in particular, the delayed simulation [13] for parity games fails to have a natural quotienting operation.

Quotienting for governed stuttering bisimulation can be done efficiently. Due to the nature of governed stuttering bisimulation, we have some freedom in the definition of the quotient, in particular when assigning vertices to players. We therefore first define a notion of minimality, and we subsequently define the quotient in terms of that notion.

Definition 6 (Minimality). *A* $\simeq$-minimal *representation of a parity game* $(V, \to, \Omega, \mathcal{P})$ *is defined as a game* $(V_{\mathrm{m}}, \to_{\mathrm{m}}, \Omega_{\mathrm{m}}, \mathcal{P}_{\mathrm{m}})$*, that satisfies the following conditions (where* $c, c', c'' \in V_{\mathrm{m}}$*):*

$$
\begin{aligned}
V_{\mathrm{m}} &= \{\, [v]_{\simeq} \mid v \in V \,\} \\
\Omega_{\mathrm{m}}(c) &= \Omega(v) \textit{ for all } v \in c \\
\mathcal{P}_{\mathrm{m}}(c) &= i, \textit{ if for all } v \in c, \textit{ and some } c' \neq c \textit{ we have } v \mathrel{{}_i\!\mapsto_{\simeq}} c' \textit{ and } v \mathrel{{}_{\neg i}\!\not\mapsto_{\simeq}} V \setminus c' \\
c \to_{\mathrm{m}} c' &\textit{ iff } v \mathrel{{}_i\!\mapsto_{\simeq}} \textit{ for all } v \in c \textit{ for some player } i \\
c \to_{\mathrm{m}} c' &\textit{ iff } v \mathrel{{}_i\!\mapsto_{\simeq}} c' \textit{ for all } v \in c \textit{ for some player } i \textit{ and } c' \neq c
\end{aligned}
$$

Observe that for the third clause above, if from some vertex v the play could be forced to c' by i without $\neg i$ having the opportunity to diverge, player i is in charge of the game when the play arrives in c. This requires the representative in the quotient to be owned by player i.

Note that a parity game may have multiple $\simeq$-minimal representations. It is not hard to verify that every parity game contains at least as many vertices and edges as its $\simeq$-minimal representations. Moreover, any parity game is governed stuttering bisimulation equivalent to all its $\simeq$-minimal representations. As a result, the governed stuttering bisimulation quotient of a graph can be defined as its least $\simeq$-minimal representation, given some arbitrary ordering on parity games. A natural ordering would be one that is induced by an ordering on players, *e.g.*, $\Box < \Diamond$.

Example 2. Consider the parity game in Figure 1b. Two of its four minimal representations are in Figure 1c and 1d. Observe that the particular player chosen for the 0 and 1 vertices is arbitrary and does not impact the solution to the games.

5.3 Governed stuttering bisimilarity refines winner equivalence

In this section, we prove that governed stuttering bisimilarity is strictly finer than winner equivalence. That is, vertices that are won by different players are never related by governed stuttering bisimilarity. In order to prove this result, we must first lift the concept of governed stuttering bisimilarity to paths.

Paths of length 1 are equivalent if the vertices they consist of are equivalent. If paths p and q are equivalent, then $p \cdot \langle v \rangle \simeq q$ iff v is equivalent to the last vertex in q, and $p \cdot \langle v \rangle \simeq q \cdot \langle w \rangle$ iff $v \simeq w$. An infinite path p is equivalent to a path q if for all finite prefixes of p there is an equivalent prefix of q and *vice versa*.

We define $\Pi^n_\varphi(v)$ to be the set of paths of length n that start in v and that are allowed by some strategy φ. $\Pi^\omega_\varphi(v)$ is then the set of all infinite paths allowed by φ, starting in v. In a similar fashion, we also define $\Psi^n_\varphi(v)$, which contains those paths starting in v that are allowed by φ and that consist of exactly n segments in which all vertices in a segment are related by $\simeq$, except the last vertex. Also included in $\Psi^n_\varphi(v)$ are infinite paths that stay in the same class forever after n or less such segments.

Definition 7 (Levels). *Formally we define the* n*th* level *paths* $\Psi^n_\varphi(v)$ *of a strategy* φ *from root vertex* v *for all paths* p *as follows:*

$$
\begin{aligned}
p \in \Psi^0_\varphi(v) \textit{ iff } & p = \langle v \rangle \\
p \cdot q \in \Psi^{n+1}_\varphi(v) \textit{ iff } & p \in \Psi^n_\varphi(v) \ \wedge \varphi \Vdash p \cdot q \ \wedge \\
& ((p \cdot q \simeq p \wedge |q| = \infty) \ \vee \\
& (\exists \bar{q}, v : q = \bar{q} \cdot \langle v \rangle \wedge p \cdot \bar{q} \simeq p \wedge p \cdot q \not\simeq p))
\end{aligned}
$$

Note that $\Pi^\omega_\varphi(v) = \Psi^\omega_\varphi(v)$. The following lemma is the basis for establishing that governed stuttering bisimilarity refines winner equivalence.

Lemma 11. *Given some* $v, w \in V$ *such that* $v \simeq w$*, then for every strategy* $\varphi \in \mathbb{S}_i$ *we have a strategy* $\psi \in \mathbb{S}^*_i$ *such that* $\forall n \in \mathbb{N} : \forall p \in \Psi^n_\psi(w) : \exists p' \in \Psi^n_\varphi(v) : p \simeq p'$

We are now in a position to prove that governed stuttering bisimilarity refines winner equivalence.

Theorem 4. *Governed stuttering bisimulation strictly refines winner equivalence.*

Proof. Let φ be a strategy for player i that wins from $v \in V$. Without loss of generality assume that φ is memoryless, and let $w \in V$ such that $v \simeq w$. By Lemma 11, we know that there is some strategy ψ such that for every path in $\Psi^\omega_\psi(w)$ there is a related path in $\Psi^\omega_\varphi(v)$. As $\Pi^\omega_\varphi(v) = \Psi^\omega_\varphi(v)$, this means that for every path starting in w that is allowed by ψ, we have an equivalent path starting in v that is allowed by φ. Equivalent paths have the same set of infinitely often recurring priorities. Any priority that may be visited infinitely often under ψ could therefore also have been visited infinitely often under φ. Therefore, ψ must be a winning strategy. The strictness of the refinement follows from, *e.g.*, the example in Figure 1.c, in which player $\Box$ wins both vertices. $\Box$

5.4 Decidability

Our algorithm for deciding governed stuttering bisimilarity is based on Groote and Vaandrager's $\mathcal{O}(nm)$ algorithm for deciding stuttering bisimilarity [15]. Before we provide the details, we introduce the necessary additional concepts.

Our algorithm requires a generalisation of the well-known notion of *attractor sets* [21] along the lines of the generalisation used for the computation of the *Until* in the alternating-time temporal logic ATL [1]. The generalisation introduces a parameter restricting the set of vertices that are considered in the attractor sets.

$$
\begin{aligned}
{}_B Attr^0_i(U) &= U \\
{}_B Attr^{n+1}_i(U) &= {}_B Attr^n_i(U) \\
&\quad \cup \{v \in B \mid \mathcal{P}(v) = i \wedge \exists v \to v' : v' \in {}_B Attr^n_i(U)\} \\
&\quad \cup \{v \in B \mid \mathcal{P}(v) \neq i \wedge \forall v \to v' : v' \in {}_B Attr^n_i(U)\} \\
{}_B Attr_i(U) &= {}_B Attr^\omega_i(U) \\
Leave_i(B, W) &= {}_B Attr_i(W) \cap B
\end{aligned}
$$

The set $Leave_i(B, W)$ captures the subset of B from which player i can force the game to $W \subseteq V$. The formal correspondence between $Leave$ and ${}_i\mapsto$ is formalised below; this allows for restating the criteria from Definition 4 in terms of $Leave$. We first capture some more general properties about $Attr$ and $Leave$.

Lemma 12. *Let P partition V, and let $B \in P$ be a block. Define $U = B \setminus {}_B Attr_{\neg i}(V \setminus B)$, then:*

1. $\forall u \in U : \mathcal{P}(u) = i \implies (\exists u \to u' : u' \in U)$
2. $\forall u \in U : \mathcal{P}(u) \neq i \implies (\forall u \to u' : u' \in U)$

Proof. We prove the two properties separately.

1. Let $u \in U$ and $\mathcal{P}(u) = i$. Towards a contradiction, suppose that $\forall u \to u'$ it holds that $u' \notin U$. Because of totality of $\to$ there must be at least one such u'. It follows that $\forall u \to u' : u' \in V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B))$. Observe that $V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B)) = {}_B Attr_{\neg i}(V \setminus B)$, hence $u' \in {}_B Attr_{\neg i}(V \setminus B)$, for all u' such that $u \to u'$. By definition of $Attr$, then also $u \in {}_B Attr_{\neg i}(V \setminus B)$, but then $u \notin B \setminus {}_B Attr_{\neg i}(V \setminus B)$, which contradicts $u \in U$, hence $(\exists u \to u' : u' \in U)$.
2. Let $u \in U$ and $\mathcal{P}(u) \neq i$. Towards a contradiction, suppose that $\exists u \to u'$ such that $u' \notin U$. Let u' be such. Observe that $u' \in V \setminus (B \setminus {}_B Attr_{\neg i}(V \setminus B))$, and thus $u' \in {}_B Attr_{\neg i}(V \setminus B)$. By definition of $Attr$, then also $u \in {}_B Attr_{\neg i}(V \setminus B)$, thus $u \notin U$, which contradicts the assumption that $u \in U$, hence $\forall u \to u' : u' \in U$. □

Lemma 13. *Let P be a partition of V, and let $B \in P$ be a block. Then for all $u \in B$: $u \; {}_i\!\!\mapsto_P$ if and only if $u \notin Leave_{\neg i}(B, V \setminus B)$.*

Proof. Let P partition V, and let $B \in P$ be a block. We first prove the implication from left to right by contraposition, *i.e.* we show that for all $u \in B$, if $u \in Leave_{\neg i}(B, V \setminus B)$, then $u \; {}_i\!\!\not\mapsto_P$. By definition of $Leave$, $Leave_{\neg i}(B, V \setminus B) = {}_B Attr_{\neg i}(V \setminus B) \cap B$. Observe that, if $u \in {}_B Attr_{\neg i}(V \setminus B) \cap B$, there is some least n such that $u \in {}_B Attr_{\neg i}(V \setminus B) \cap B$. We show by induction on n that $\forall u \in B : u \in {}_B Attr^n_{\neg i}(V \setminus B) \cap B \implies u \; {}_i\!\!\not\mapsto_P$.

- $n = 0$. We find that ${}_B Attr^0_{\neg i}(V \setminus B) = V \setminus B$, and $(V \setminus B) \cap B = \emptyset$, hence the statement vacuously holds.
- $n = m + 1$. As induction hypothesis asssume that $\forall u \in B : u \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies u \; {}_i\!\!\not\mapsto_P$.
  Let $u \in {}_B Attr^{m+1}_{\neg i}(V \setminus B) \cap B$. By definition of $Attr$, we find three cases.
  - $u \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B$. This follows immediately from the induction hypothesis.
  - $u \in \{v \in B \mid \mathcal{P}(v) \neq i \wedge \exists v \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)\} \cap B$. So we know that $\mathcal{P}(u) \neq i$, and $\exists u \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)$. Let u' be such, and observe that either $u' \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B$ or $u' \in V \setminus B$ because $(V \setminus B) \subseteq {}_B Attr^m_{\neg i}(V \setminus B)$. In the latter case there is a strategy for player $\neg i$ such that $u \; {}_{\neg i}\!\!\mapsto_P V \setminus B$, and by Lemma 3 we find $u \; {}_i\!\!\not\mapsto_P$. In the first case, observe that by the induction hypothesis, $u' \; {}_i\!\!\not\mapsto_P$. According Lemma 3, we find that $u' \; {}_{\neg i}\!\!\mapsto_P V \setminus B$. Application of Lemma 4 gives us that $u \; {}_{\neg i}\!\!\mapsto_P V \setminus B$, and again using Lemma 3, we have $u \; {}_i\!\!\not\mapsto_P$.
  - $u \in \{v \in B \mid \mathcal{P}(v) = i \wedge \forall v \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)\} \cap B$. So we know that $\mathcal{P}(u) = i$, and $\forall u \to v' : v' \in {}_B Attr^m_{\neg i}(V \setminus B)$. By totality of $\to$ there is at least one such v'. Observe that $u \; {}_{\neg i}\!\!\mapsto_P {}_B Attr^m_{\neg i}(V \setminus B)$. According to the induction hypothesis, we know $\forall v \in B : v \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies v \; {}_i\!\!\not\mapsto_P$. Applying Lemma 3 we also know that $\forall v \in B : v \in {}_B Attr^m_{\neg i}(V \setminus B) \cap B \implies v \; {}_{\neg i}\!\!\mapsto_P (V \setminus B)$. Using Lemma 4 we find that $u \; {}_{\neg i}\!\!\mapsto_P (V \setminus B)$. Another application of Lemma 3 gives us the desired result $u \; {}_i\!\!\not\mapsto_P$.

Finally we prove the implication from right to left, *i.e.* we prove that $\forall u \in B : u \notin Leave_{\neg_i}(B, V \setminus B) \implies u \,{}_i\!\mapsto_P$. Define $U \subseteq B$ to be the subset of B that cannot be forced by player $\neg i$ to leave B, *i.e.* $U = B \setminus Leave_{\neg_i}(B, V \setminus B) = B \setminus {}_{\neg i}Attr_B(V \setminus B)$. Observe that for all $u \in V : u \in U$ iff $u \notin Leave_{\neg_i}(B, V \setminus B)$, so we reformulate our goal as $\forall u \in B : u \in U \implies u \,{}_i\!\mapsto_P$. Assuming a total ordering on vertices of U, we define strategy $\sigma \in \mathbb{S}_i$ that is defined for vertices in U, such that

$$\sigma(u) = \min\{u' \in U \mid u \to u'\}$$

Observe that $\{u' \in U \mid u \to u'\} \neq \emptyset$ due to Lemma 12, hence $\sigma(u)$ is well-defined. Furthermore for all $v \in U$ with $\mathcal{P}(v) \neq i$, all successors are in U due to Lemma 12. As a result, $u \,{}_\sigma\!\mapsto_P$, and hence $u \,{}_i\!\mapsto_P$. □

Lemma 14. *Let P partition V, and let $B, B' \in P$ such that $B \neq B'$. Let $v \in B$ such that $v \to B'$. Then for all $w \in B$ it holds that $w \,{}_{\mathcal{P}(v)}\!\mapsto_P B'$ if and only if $w \in Leave_{\mathcal{P}(v)}(B, B')$.*

The proof of this lemma follows the same line of reasoning as the proof of Lemma 13.

Groote and Vaandrager's algorithm for stuttering bisimulation repeatedly refines a carefully chosen initial partition P_0 using a so-called *splitter*. We apply the same principle, choosing P_0 such that for all $v, v' \in V$, $v \; P_0 \; v'$ if and only if $\Omega(v) = \Omega(v')$ as our initial partition. As our splitter, we define a function pos that returns the set of vertices in B from which a given player i can force the play to reach B', or, in case $B = B'$, force the play to diverge:

$$pos_i(B, B') = \begin{cases} \{v \in B \mid v \,{}_i\!\mapsto_P\} & \text{if } B = B' \\ \{v \in B \mid v \,{}_i\!\mapsto_P B'\} & \text{if } B \neq B' \end{cases}$$

In line with [15], we say that B' is a *splitter* of B if and only if $\emptyset \neq pos_i(B, B') \neq B$ for some player i. A partition P is *stable with respect to a block* $B \in P$ if B is not a splitter of any block in P. The partition itself is stable if it is stable with respect to all its blocks.

A high-level description of our algorithm for governed stuttering bisimulation, is given as Algorithm 1. We now detail Algorithm 1 using the approach from [15]. Given the nature of the definition of divergence in governed stuttering bisimulation, we cannot remove divergent states in a preprocessing step.

Our algorithm maintains a partition P. Initially $P = P_0$, such that for all v, v': $v \; P_0 \; v'$ iff $\Omega(v) = \Omega(v')$. Algorithm 2 then computes the largest governed stuttering bisimulation.

Given a parity game, our algorithm maintains two lists of blocks, $todo$ and $stable$. A block B' is in $stable$ if the current partition is stable with respect to B', otherwise B' is in $todo$. Initially all blocks in P_0 are in $todo$. While the todo list is not empty, we check for each block B whether it is a splitter of any block B' using the routine $TrySplit(B, B')$. If a splitter is found, the todo list is updated accordingly, and if some inert transition has become non-inert all stable blocks are added to the todo list in accordance with Lemma 16.

Algorithm 1 Decision procedure for $\simeq$

$n \leftarrow 0$
repeat
  $splitter \leftarrow \bot$
  for each $B \in P_n$ and player i **do** { Find splitter in $\mathcal{O}(nm)$ }
    if there exists $v \in B$ with $v \rightarrow B'$ and $\emptyset \neq pos_i(B, B') \neq B$ for $B' \in P_n$ **then**
      $splitter \leftarrow (B, pos_i(B, B'))$
    end if
  end for
  if $splitter = (B, Pos)$ **then** { Refine partition in $\mathcal{O}(m)$ }
    $P_{n+1} \leftarrow (P_n \setminus \{B\}) \cup \{Pos, B \setminus Pos\}$
  end if
  $n \leftarrow n + 1$
until $P_{n-1} = P_n$

Algorithm 2 Algorithm for computing $\simeq$

$todo \leftarrow P_0$; $stable \leftarrow \emptyset$;
repeat
  $B' \leftarrow head(todo)$
  for each $v \rightarrow v' \in B'.incoming$ **do**
    if $v \notin B$ **then**
      $BL.append(v)$
    end if
  end for
  repeat
    $B \leftarrow BL.pop()$ {Determine for each block B whether B' is a splitter for B}
    $(foundsplitter, inert\_becomes\_non\_inert, B_1, B_2) \leftarrow TrySplit(B, B')$
  until $foundsplitter$ or $BL = \emptyset$
  if $foundsplitter$ **then**
    $todo.remove(B)$
    $todo.append(B_1, B_2)$
    if $inert\_becomes\_non\_inert$ **then**
      $todo \leftarrow todo + stable$; $stable \leftarrow \emptyset$
    end if
  else
    $todo.remove(B')$
    $stable.append(B')$
  end if
until $todo = \emptyset$

We next elaborate on $TrySplit(B, B')$, which is given as Algorithm 3. This determines whether B' is a splitter of B, using $Leave$ according to Lemmata 13 and 14. If a splitter is found, the actual splitting is performed; the non-inert edges are added to the appropriate block, and for all inert edges it is checked whether they are still inert, and if not they are also added as non-inert edges to the appropriate block.

The following lemma shows that our algorithm indeed computes $\simeq$.

Algorithm 3 $TrySplit(B, B')$

```
foundsplitter ← false; inert_becomes_non_inert ← false
i ← 0
repeat
    B_1 ← pos_i(B, B')
    i ← i + 1
    foundsplitter ← ∅ ≠ B_1 ≠ B
until i > 1 or foundsplitter
if foundsplitter then
    B_2 ← B \ B_1
    for v → v' ∈ B.incoming do
        if v' ∈ B_1 then
            B_1.incoming.append(v → v')
        else
            B_2.incoming.append(v → v')
        end if
    end for
    for C ∈ {B_1, B_2} do
        for v ∈ C do
            for v' → v ∈ v.incoming do
                if v' ∉ C then
                    inert_becomes_non_inert ← true
                    v.incoming.remove(v' → v)
                    C.incoming.append(v' → v)
                end if
            end for
        end for
    end for
    return (foundsplitter, inert_becomes_non_inert, B_1, B_2)
else
    return (foundsplitter, inert_becomes_non_inert, ∅, ∅)
end if
```

Lemma 15. *If P is a stable partition refining P_0, then P is a governed stuttering bisimulation. If P is the largest such partition then P coincides with $\simeq$.*

Proof. We first prove the first part of the statement. Let P be a stable partition refining P_0, and let $v, v' \in V$ such that $v \ P \ v'$. We show that P satisfies the three properties described in Theorem 1. $\Omega(v) = \Omega(v')$ follows immediately as P is a refinement of P_0. The transfer and divergence properties follow immediately from the observation that P is stable, and the definition of pos.

For the second part of the statement observe that $\simeq$ is an equivalence relation according to Theorem 3. Furthermore it is defined to be the largest governed stuttering bisimulation, and it induces a stable refinement of P_0. As any stable refinement is a governed stuttering bisimulation our result follows.

That stable blocks only have to be reconsidered if inert transitions become non-inert follows from the following lemma.

Lemma 16. *Let P_n and P_{n+1} be partitions of V, such that P_n and P_{n+1} have the same inert transitions. Assume that P_{n+1} refines P_n. Let $B' \in P_n, P_{n+1}$ such that P_n is stable with respect to B'. Then also P_{n+1} is stable with respect to B'.*

Proof. Towards a contradiction, suppose that there exists a block $B \in P_{n+1}$ such that $B' \in P_n, P_{n+1}$ is a splitter of B, we show that than P_n is not stable with respect to B'. We distinguish two cases.

1. $B = B'$. As B' is a splitter of itself under P_{n+1}, and $B' \in P_n$, we immediately find that B' is a splitter of itself under P_n, which is a contradiction.
2. $B \neq B'$. As B' is a splitter of B, we know that $v, v' \in B$ such that $v \;{}_i\!\!\mapsto_{P_{n+1}} B' \wedge v' \;{}_i\!\!\not\mapsto_{P_{n+1}} B'$ for some player i. Let v, v' be such. Observe that there is a block $B'' \in P_n$ such that $B \subseteq B''$. In case $B = B''$, then immediately we find that B' is a splitter of B'', which violates the stability of P_n. Suppose $B \subset B''$. As P_n and P_{n+1} have the same inert transitions, there are no $u \in B$, $u' \in B''$ such that $u \to u'$. From this, and the assumption that $v' \;{}_i\!\!\not\mapsto_{P_{n+1}} B'$ it follows that $v' \;{}_i\!\!\not\mapsto_{P_n} B'$. Likewise we have that $v \;{}_i\!\!\mapsto_{P_n} B'$, and hence B' is a splitter of B'' under P_n, which contradicts stability of P_n.

In both cases we arrive at a contradiction, hence P_{n+1} is stable with respect to B'.

The number of iterations the algorithm requires to compute $\simeq$ is bounded in the following theorem.

Theorem 5. *Algorithm 2 terminates after at most $n - |P_0|$ refinement steps. The resulting partition P_f is the coarsest stable partition refining P_0.*

Proof. Termination of the algorithm after at most $n - |P_0|$ refinement steps is straightforward. Next we show that the resulting partition P_f is the coarsest stable partition refining P_0. We prove, by induction on the number of refinement steps j, that any stable partition refining P_0 is also a refinement of the current partition P_j. Clearly the statement holds initially. Let R be a stable refinement of P_0. By the induction hypothesis, R is a refinement of P_j. Let P_{j+1} be the refinement of P_j, after refining using splitting pair (B, B') (*i.e.* B' is a splitter of B). We show that R is also a refinement of P_{j+1}. Let C be a block in R. Then there is a block D in P_j such that $C \subseteq D$. We show that C is included in a block of P_{j+1}. In case $D \neq B$ we are done. In case $D = B$, assume that splitting was done with respect to player i, then we have to show that either $C \subseteq pos_i(B, B')$, or $C \subseteq B \setminus pos_i(B, B')$.

Towards a contradiction, suppose that there are $v, v' \in C$ such that $v \in pos_i(B, B')$ and $v' \notin pos_i(B, B')$. We distinguish two cases:

- $B \neq B'$. As $v \in pos_i(B, B')$, we know $v \;{}_i\!\!\mapsto_{P_j} B'$. We know there is a sequence of classes $C_1, \ldots, C_n$ in R, with $C_1 = C$, $C_n = C'$, and $C_j \;{}_i\!\!\to C_{j+1}$, in other words, in each of the classes C_j, the game can be forced to C_{j+1} by player i. Since R is a stable refinement of P_0, and $v, v' \in C$ we find that also $v' \in pos_i(B, B')$, which is a contradiction.
- $B = B'$. As $v \in pos_i(B, B')$, we know $v \;{}_i\!\!\mapsto_{P_j}$. We know there is a sequence of classes $C_1, \ldots, C_n$ in R such that $C_i \;{}_i\!\!\mapsto_R C_{i+1}$, and $C = C_1 = C_n$. Following a similar line of reasoning as in the previous case, we find that then $v' \in pos_i(B, B')$, which is a contradiction.

Given the above considerations, we can determine the running time complexity of our algorithm as follows.

Theorem 6. *Algorithm 1 decides $\simeq$ in $\mathcal{O}(n^2m)$ time for a parity game that contains n vertices and m edges.*

Proof. Deciding whether a block B' is a splitter of block B in Algorithm 3 takes $\mathcal{O}(m_B + n_B)$ time (the time required to compute the attractor set). If a splitter is found, the actual splitting takes $\mathcal{O}(m)$ time. As a result, deciding whether a block B' is a splitter for the current partition takes $\Sigma_B(\mathcal{O}(m_B + n_B)) = \mathcal{O}(m)$ time. Finding a splitter of the current partition (if it exists) hence has time complexity $\mathcal{O}(nm)$. As only $n - |P_0|$ refinement steps are possible (Lemma 5), the time complexity of $\mathcal{O}(n^2m)$ follows.

Our time complexity is worse than the $\mathcal{O}(nm)$ achieved by the original algorithm for deciding stuttering bisimulation. The extra factor $\mathcal{O}(n)$ is due to the complexity required to search for a splitter which, in our case, requires $\mathcal{O}(nm)$ time, instead of the original $\mathcal{O}(m)$ time.

The hard case turns out to be the problem of finding a splitter for a block B that consists of a single strongly connected component in which all vertices in B are divergent for exactly one player. For this problem, we only have an $\mathcal{O}(nm_C)$ algorithm, leading to the $\mathcal{O}(nm)$ time bound for a single iteration. The following problem statement formalises this 'hard case'.

Problem 1. Let P be a partition, and let $C \in P$ be a block, such that C is a strongly connected component, and for all $v \in C$ we have $v \,{}_i\!\mapsto_P$, and $v \,{}_{\neg i}\!\not\mapsto_P$. Determine, in $\mathcal{O}(m)$ time, whether there exist $v, v' \in C$ and a block $B \in P$ such that $v \,{}_i\!\mapsto_P B$ and $v' \,{}_i\!\not\mapsto_P B$.

Claim. Given a solution to Problem 1, the largest governed stuttering bisimulation can be computed in $\mathcal{O}(nm)$ time.

6 Experiments

While the running time of our algorithm for governed stuttering bisimilarity is theoretically worse than that of the algorithm for stuttering bisimilarity, we expect that for solving parity games, in practice both are comparable. We test this hypothesis on a set of over 200 real-life model checking problems, part of which was previously used to study the effect of stuttering bisimilarity for parity games, see [9].

Whereas in [9] a signature based approach [5] was used, in the present paper we use the Groote-Vaandrager algorithm for computing stuttering bisimilarity in order to answer our hypothesis. For computing governed stuttering bisimilarity we have modified the implementation of Groote-Vaandrager to include the changes presented in Algorithm 1.

For running our experiments we reuse the setup of [9] for solving parity games, *i.e.*, we use an optimised C++ implementation of the *small progress measures* algorithm [17], and the optimised variants of the small progress measures, recursive [21, 25] and *bigstep* algorithms [23] offered by the PGSolver [11] toolset.

All experiments were conducted on a machine consisting of 28 Intel® Xeon© E5520 Processors running at 2.27GHz, with 1TB of shared main memory, running a 64-bit Linux distribution using kernel version 2.6.27. None of our experiments employ multi-core features.

6.1 Test sets

The parity games that were used for our experiments are clustered into four test sets. We give a brief description of each of the sets below.

Model checking Our main interest is in the practical implications of governed stuttering bisimilarity reduction on solving model checking problems. To this end, a number of model checking problems have been selected from the literature [20, 3, 14]. The properties that have been checked include fairness, liveness and safety properties.

Games The second test set considers a number of turn-based, two player board games. For each of these games, and for each player, we have encoded the property that said player can play the game in such a way that, regardless of the play of the opponent, she can win the game.

PGSolver The third test set was taken from [11] and consists of the elevator problem and the Hanoi towers problem described in that paper. It also includes alternative encodings of these problems, taken from [9].

Equivalence checking The last test set consists of a number of equivalence checking problems encoded into parity games as described in [8].

The problems in these test sets are scalable. In every test set, a number of instances of every problem is included. Each problem gives rise to a parity game with at most 4 different priorities, which is typical for practical verification problems.

The model checking, PGSolver and equivalence checking problems were studied before in the setting of stuttering bisimilarity [9]. We extended that test set to include more examples of parity games with alternations between players and priorities. We can expect improved reductions compared to stuttering bisimilarity in these cases.

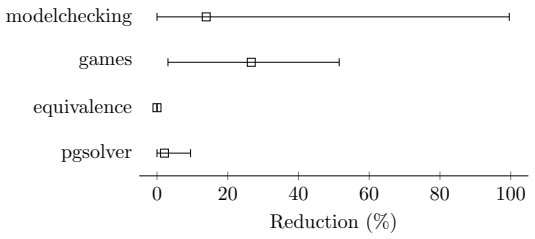
6.2 Measurements: Size and Time

To analyse the performance of our reduction, we measured the difference in the sizes (computed as the sum of the number of vertices and the number of edges) of the stuttering and governed stuttering minimal games. A reduction of 0% means that the governed stuttering bisimilarity reduced game has the same size as the stuttering bisimilarity reduced game.

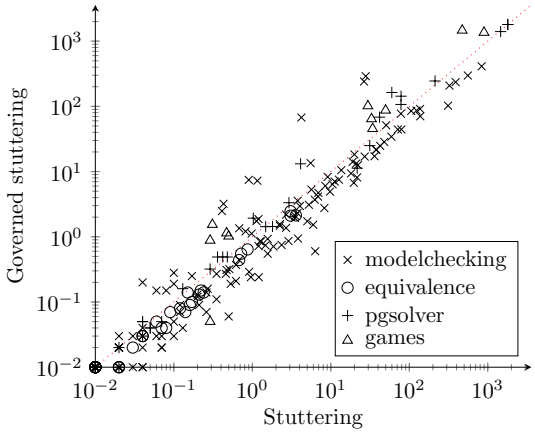
For every problem in the test set, we compute the reduction as the average reduction over all instances of that problem. We do this in order to measure the reduction rate for the different problems, rather than for the instances. Figure 2a shows the average reduction for problems in each test set, together with the minimal and maximal reduction achieved within that set.

In addition, we measured the times needed to reduce the parity games plus the time needed to solve the reduced game using the fastest solver. That is, the sum of these

two is the *total solving time* for a parity game. This way, our results can be compared to those listed in [9]. In Figure 2b, every data point represents a problem instance, of which the total solving time of the stuttering minimal game determines the x-coordinate, and the total solving time for the governed stuttering bisimilarity minimal game determines the y-coordinate.



(a) Minimum, maximum and average reduction of parity games using governed stuttering bisimulation reduction, as percentage of the size after stuttering bisimilarity reduction



(b) Solving times (=sum of reduction time and subsequent solving) in seconds using governed stuttering bisimilarity set out against the solving times using stuttering bisimilarity. The dotted line is defined as $x = y$ and serves as a reference. Note that the axes are in log scale.

Fig. 2: Comparison of sizes and times of reductions

6.3 Discussion

Figure 2b shows that solving times for governed stuttering bisimilarity are generally comparable to those for stuttering bisimilarity, confirming our hypothesis.

Whether governed stuttering bisimilarity offers additional reductions over stuttering bisimilarity depends largely on the kind of property that is checked, and the resulting structure of the parity game. On average, a modest additional reduction is achieved, and there are practical cases in which the additional reduction is almost 100%.

For several model checking cases, stuttering bisimilarity already reduces the parity game to a graph with one vertex per priority. Obviously, governed stuttering bisimilarity cannot improve on that. However, in one of our problems (model checking a leadership

protocol) a reduction of almost 100% is achieved, increasing the average reduction for this test set.

The properties that we considered on two-player games naturally give rise to alternations between players in the parity game. For these type of properties, the reduction achieved using governed stuttering bisimilarity surpasses that of stuttering bisimilarity by about 20% on average. Similar results for parity games obtained for controller synthesis (see *e.g.* [2]) may be obtained as these exhibit similar structures.

For the equivalence cases, stuttering bisimilarity reduction already yields games of a small size and governed stuttering bisimilarity does not reduce any further.[1]

Interestingly, one of the PGSolver cases taken from [11] shows a better reduction using governed stuttering bisimilarity, in contrast to an alternative encoding also used in [9].

Summarising, we conclude that governed stuttering bisimilarity reduces slightly better than stuttering bisimilarity, without noticable loss of performance.

7 Related work

As observed in Fritz' thesis [12], *direct simulation* for parity games led to disappointing reductions, spurring Fritz and Wilke to investigate a weaker notion, called *delayed simulation* [13] and its induced equivalence. Delayed simulation equivalence is incomparable to governed stuttering bisimilarity. Contrary to governed stuttering bisimilarity, delayed simulation equivalence has the capability to relate vertices with different priorities. On the other hand, governed stuttering bisimilarity can relate vertices with the same priority in cases that delayed simulation equivalence cannot, as illustrated by the two parity games below, in which governed stuttering bisimulation relates all vertices with equal priority whereas delayed simulation equivalence does not:

Contrary to governed stuttering bisimulation, the definition of the simulation relation is entirely in terms of a *simulation game*, *viz.*, a game graph equipped with Büchi winning conditions. The simulation game gives rise to an $\mathcal{O}(d^2n^3m)$ algorithm for deciding delayed simulation (here, n is the number of vertices, m the number of edges, and d the number of different priorities in the game), significantly exceeding our $\mathcal{O}(n^2m)$ complexity for governed stuttering bisimulation.

Apart from delayed simulation, in the setting of Boolean equation systems, the notion of *idempotence-identifying bisimilarity* was defined and investigated [19]. This equivalence relation enables one to relate conjunctive equations to disjunctive equations. In parity games, this translates to being able to relate $\Box$ vertices and $\Diamond$ vertices, respectively. Idempotence-identifying bisimilarity is much finer than governed stuttering bisimilarity, as the former is based on strong bisimilarity. Interestingly, the complexity of deciding idempotence-identifying bisimilarity is the same as for strong bisimilarity.

[1] [9] reports a poor reduction for stuttering equivalence in these cases. This was caused by "optimisations" that were used during generation of the parity games.

8 Concluding remarks

We have described a non-trivial modification of stuttering bisimulation that allows relating vertices that belong to different players. The resulting relation, dubbed *governed stuttering bisimulation*, is an equivalence relation that can be decided in $\mathcal{O}(n^2m)$ time using a partition refinement algorithm. Although this complexity is worse than the $\mathcal{O}(nm)$ time complexity for deciding stuttering bisimulation, our experiments indicate that this factor does not manifest itself in practice. In fact, the algorithm is largely competitive with the one for stuttering bisimilarity.

An obvious question is whether elements of Fritz and Wilke's delayed simulation [13] and governed stuttering bisimulation can be combined. Given the complexity of the proofs of most of our results for governed stuttering bisimulation and our attempts to weakening governed stuttering bisimulation along these lines, we are rather sceptic about the chances of success. Even if one would manage to define such a relation, it would likely have little practical significance due to the prohibitive complexity of delayed simulation.

An interesting extension of our work could be to generalise the concepts of governed stuttering bisimilarity to games with other payoff functions that are insensitive to stuttering. We expect such a generalisation to be reasonably straightforward.

Finally, we observe that stuttering bisimulation (also known as *branching bisimulation* in labelled transition systems) underlies several confluence reduction techniques for syntactic system descriptions, see [6]. Such reductions partly side-step the state-space explosion. We believe that our study offers the required foundations for bringing similar-spirited confluence reduction techniques to a setting of symbolic representations of parity games.

References

1. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, September 2002.
2. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *TCS*, 303(1):7–34, 2003.
3. B. Badban, W. Fokkink, J.F. Groote, J. Pang, and J. v.d. Pol. Verification of a sliding window protocol in μCRL and PVS. *FAC*, 17:342–388, 2005.
4. D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *STACS'06*, volume 3884 of *LNCS*, pages 436–524. Springer, 2006.
5. S.C.C. Blom and S. Orzan. Distributed branching bisimulation reduction of state spaces. *Electronic Notes in Theoretical Computer Science*, 89(1):99–113, 2003.
6. S.C.C. Blom and J.C. v.d. Pol. State space reduction by proving confluence. In *CAV'02*, volume 2404 of *LNCS*, pages 676–694. Springer, 2002.
7. M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *TCS*, 59:115–131, 1988.
8. T. Chen, B. Ploeger, J. v.d. Pol, and T.A.C. Willemse. Equivalence Checking for Infinite Systems Using Parameterized Boolean Equation Systems. In *CONCUR'07*, pages 120–135, 2007.
9. S. Cranen, J.J.A. Keiren, and T.A.C. Willemse. Stuttering mostly speeds up solving parity games. In *NFM'11*, volume 6617 of *LNCS*, pages 207–221. Springer, 2011.

10. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society.
11. O. Friedmann and M. Lange. Solving parity games in practice. In *ATVA'09*, volume 5799 of *LNCS*, pages 182–196. Springer, 2009.
12. C. Fritz. *Simulation-Based Simplification of omega-Automata*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2005.
13. C. Fritz and T. Wilke. Simulation relations for alternating parity automata and parity games. In *DLT'06*, volume 4036 of *LNCS*, pages 59–70. Springer, 2006.
14. J.F. Groote, J. Pang, and A.G. Wouters. Analysis of a distributed system for lifting trucks. In *JLAP*, volume 55, pages 21–56. Elsevier, 2003.
15. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *ICALP'90*, volume 443 of *LNCS*, pages 626–638. Springer, 1990.
16. M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *IPL*, 68(3):119–124, 1998.
17. M. Jurdziński. Small progress measures for solving parity games. In *STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.
18. M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. In *SODA'06*, pages 117–123. ACM/SIAM, 2006.
19. J.J.A. Keiren and T.A.C. Willemse. Bisimulation Minimisations for Boolean Equation Systems. In *HVC'09*, volume 6405 of *LNCS*, 2011.
20. S.P. Luttik. Description and formal specification of the link layer of P1394. In *Workshop on Applied Formal Methods in System Design*, pages 43–56, 1997.
21. R. McNaughton. Infinite games played on finite graphs. *APAL*, 65(2):149–184, 1993.
22. J. Obdrzálek. Clique-Width and Parity Games. In *CSL*, pages 54–68, 2007.
23. S. Schewe. Solving parity games in big steps. In *FSTTCS'07*, volume 4855 of *LNCS*, pages 449–460. Springer, 2007.
24. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, May 1996.
25. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1-2):135 – 183, 1998.

Science Reports **Department of Mathematics and Computer Science**
Technische Universiteit Eindhoven

If you want to receive reports, send an email to: wsinsan@tue.nl (we cannot guarantee the availability of the requested reports).

In this series appeared (from 2009):

09/01	Wil M.P. van der Aalst, Kees M. van Hee, Peter Massuthe, Natalia Sidorova and Jan Martijn van der Werf	Compositional Service Trees
09/02	P.J.l. Cuijpers, F.A.J. Koenders, M.G.P. Pustjens, B.A.G. Senders, P.J.A. van Tilburg, P. Verduin	Queue merge: a Binary Operator for Modeling Queueing Behavior
09/03	Maarten G. Meulen, Frank P.M. Stappers and Tim A.C. Willemse	Breadth-Bounded Model Checking
09/04	Muhammad Atif and MohammadReza Mousavi	Formal Specification and Analysis of Accelerated Heartbeat Protocols
09/05	Michael Franssen	Placeholder Calculus for First-Order logic
09/06	Daniel Trivellato, Fred Spiessens, Nicola Zannone and Sandro Etalle	POLIPO: Policies & OntoLogies for the Interoperability, Portability, and autOnomy
09/07	Marco Zapletal, Wil M.P. van der Aalst, Nick Russell, Philipp Liegl and Hannes Werthner	Pattern-based Analysis of Windows Workflow
09/08	Mike Holenderski, Reinder J. Bril and Johan J. Lukkien	Swift mode changes in memory constrained real-time systems
09/09	Dragan Bošnački, Aad Mathijssen and Yaroslav S. Usenko	Behavioural analysis of an I²C Linux Driver
09/10	Ugur Keskin	In-Vehicle Communication Networks: A Literature Survey
09/11	Bas Ploeger	Analysis of ACS using mCRL2
09/12	Wolfgang Boehmer, Christoph Brandt and Jan Friso Groote	Evaluation of a Business Continuity Plan using Process Algebra and Modal Logic
09/13	Luca Aceto, Anna Ingolfsdottir, MohammadReza Mousavi and Michel A. Reniers	A Rule Format for Unit Elements
09/14	Maja Pešić, Dragan Bošnački and Wil M.P. van der Aalst	Enacting Declarative Languages using LTL: Avoiding Errors and Improving Performance
09/15	MohammadReza Mousavi and Emil Sekerinski, Editors	Proceedings of Formal Methods 2009 Doctoral Symposium
09/16	Muhammad Atif	Formal Analysis of Consensus Protocols in Asynchronous Distributed Systems
09/17	Jeroen Keiren and Tim A.C. Willemse	Bisimulation Minimisations for Boolean Equation Systems
09/18	Kees van Hee, Jan Hidders, Geert-Jan Houben, Jan Paredaens, Philippe Thiran	On-the-fly Auditing of Business Processes
10/01	Ammar Osaiweran, Marcel Boosten, MohammadReza Mousavi	Analytical Software Design: Introduction and Industrial Experience Report
10/02	F.E.J. Kruseman Aretz	Design and correctness proof of an emulation of the floating-point operations of the Electrologica X8. A case study

10/03	Luca Aceto, Matteo Cimini, Anna Ingolfsdottir, MohammadReza Mousavi and Michel A. Reniers	On Rule Formats for Zero and Unit Elements
10/04	Hamid Reza Asaadi, Ramtin Khosravi, MohammadReza Mousavi, Neda Noroozi	Towards Model-Based Testing of Electronic Funds Transfer Systems
10/05	Reinder J. Bril, Uğur Keskin, Moris Behnam, Thomas Nolte	Schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks revisited
10/06	Zvezdan Protić	Locally unique labeling of model elements for state-based model differences
10/07	C.G.U. Okwudire and R.J. Bril	Converting existing analysis to the EDP resource model
10/08	Muhammed Atif, Sjoerd Cranen, MohammadReza Mousavi	Reconstruction and verification of group membership protocols
10/09	Sjoerd Cranen, Jan Friso Groote, Michel Reniers	A linear translation from LTL to the first-order modal μ-calculus
10/10	Mike Holenderski, Wim Cools Reinder J. Bril, Johan J. Lukkien	Extending an Open-source Real-time Operating System with Hierarchical Scheduling
10/11	Eric van Wyk and Steffen Zschaler	1st Doctoral Symposium of the International Conference on Software Language Engineering (SLE)
10/12	Pre-Proceedings	3rd International Software Language Engineering Conference
10/13	Faisal Kamiran, Toon Calders and Mykola Pechenizkiy	Discrimination Aware Decision Tree Learning
10/14	J.F. Groote, T.W.D.M. Kouters and A.A.H. Osaiweran	Specification Guidelines to avoid the State Space Explosion Problem
10/15	Daniel Trivellato, Nicola Zannone and Sandro Etalle	GEM: a Distributed Goal Evaluation Algorithm for Trust Management
10/16	L. Aceto, M. Cimini, A.Ingolfsdottir, M.R. Mousavi and M. A. Reniers	Rule Formats for Distributivity
10/17	L. Aceto, A. Birgisson, A. Ingolfsdottir, and M.R. Mousavi	Decompositional Reasoning about the History of Parallel Processes
10/18	P.D. Mosses, M.R. Mousavi and M.A. Reniers	Robustness os Behavioral Equivalence on Open Terms
10/19	Harsh Beohar and Pieter Cuijpers	Desynchronisability of (partial) closed loop systems
11/01	Kees M. van Hee, Natalia Sidorova and Jan Martijn van der Werf	Refinement of Synchronizable Places with Multi-workflow Nets - Weak termination preserved!
11/02	M.F. van Amstel, M.G.J. van den Brand and L.J.P. Engelen	Using a DSL and Fine-grained Model Transformations to Explore the boundaries of Model Verification
11/03	H.R. Mahrooghi and M.R. Mousavi	Reconciling Operational and Epistemic Approaches to the Formal Analysis of Crypto-Based Security Protocols
11/04	J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius	Benefits of Applying Formal Methods to Industrial Control Software
11/05	Jan Friso Groote and Jan Lanik	Semantics, bisimulation and congruence results for a general stochastic process operator
11/06	P.J.L. Cuijpers	Moore-Smith theory for Uniform Spaces through Asymptotic Equivalence
11/07	F.P.M. Stappers, M.A. Reniers and S. Weber	Transforming SOS Specifications to Linear Processes
11/08	Debjyoti Bera, Kees M. van Hee, Michiel van Osch and Jan Martijn van der Werf	A Component Framework where Port Compatibility Implies Weak Termination
11/09	Tseesuren Batsuuri, Reinder J. Bril and Johan Lukkien	Model, analysis, and improvements for inter-vehicle communication using one-hop periodic broadcasting based on the 802.11p protocol

11/10	Neda Noroozi, Ramtin Khosravi, MohammadReza Mousavi and Tim A.C. Willemse	Synchronizing Asynchronous Conformance Testing
11/11	Jeroen J.A. Keiren and Michel A. Reniers	Type checking mCRL2
11/12	Muhammad Atif, MohammadReza Mousavi and Ammar Osaiweran	Formal Verification of Unreliable Failure Detectors in Partially Synchronous Systems
11/13	J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius	Experience report on developing the Front-end Client unit under the control of formal methods
11/14	J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius	Ananlyzing a Controller of a Power Distribution Unit Using Formal Methods
11/15	John Businge, Alexander Serebrenik and Mark van den Brand	Eclipse API Usage: The Good and The Bad
11/16	J.F. Groote, A.A.H. Osaiweran, M.T.W. Schuts and J.H. Wesselius	Investigating the Effects of Designing Control Software using Push and Poll Strategies
11/17	M.F. van Amstel, A. Serebrenik And M.G.J. van den Brand	Visualizing Traceability in Model Transformation Compositions
11/18	F.P.M. Stappers, M.A. Reniers, J.F. Groote and S. Weber	Dogfooding the Structural Operational Semantics of mCRL2
12/01	S. Cranen	Model checking the FlexRay startup phase
12/02	U. Khadim and P.J.L. Cuijpers	Appendix C / G of the paper: Repairing Time-Determinism in the Process Algebra for Hybrid Systems ACP
12/03	M.M.H.P. van den Heuvel, P.J.L. Cuijpers, J.J. Lukkien and N.W. Fisher	Revised budget allocations for fixed-priority-scheduled periodic resources
12/04	Ammar Osaiweran, Tom Fransen, Jan Friso Groote and Bart van Rijnsoever	Experience Report on Designing and Developing Control Components using Formal Methods
12/05	Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse	A cure for stuttering parity games