International Journal on Software Tools for Technology Transfer (2019) 21:325–349
https://doi.org/10.1007/s10009-019-00509-3

SPIN 2017

An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space

John Fearnley[1] · Sanjay Jain[2] · Bart de Keijzer[3] · Sven Schewe[1] · Frank Stephan[2,4] · Dominik Wojtczak[1]

Published online: 25 February 2019

Abstract
Parity games play an important role in model checking and synthesis. In their paper, Calude et al. have recently shown that these games can be solved in quasi-polynomial time. We show that their algorithm can be implemented efficiently: we use their data structure as a progress measure, allowing for a backward implementation instead of a complete unravelling of the game. To achieve this, a number of changes have to be made to their techniques, where the main one is to add power to the antagonistic player that allows for determining her rational move without changing the outcome of the game. We provide a first implementation for a quasi-polynomial algorithm, test it on small examples, and provide a number of side results, including minor algorithmic improvements, a quasi-bi-linear complexity in the number of states and edges for a fixed number of colours, matching lower bounds for the algorithm of Calude et al., and a complexity index associated to our approach, which we compare to the recently proposed register index.

Keywords Parity games · Progress measure · Quasi-polynomial

1 Introduction

Parity games are two-player zero-sum games played on a finite graph. The two players, named *Even* and *Odd*, move a token around the graph until a cycle is formed. Each vertex is labelled with an integer *colour*, and the winner is determined by the *parity* of the largest colour that appears on the cycle: player *Even* wins if it is an even colour, and player *odd* wins otherwise.

Parity games have been the focus of intense study [3–7,10,11,13,20–22,25,27–29,31,34–36,39,41], in part due to their practical applications. Solving parity games is the central and most expensive step in many model checking [1,8,9,11,24,40], satisfiability checking [24,33,38,40], and synthesis [19,30,32] algorithms.

Parity games have also attracted attention due to their unusual complexity status. The problem of determining the winner of a parity game is known to lie in UP ∩ co-UP [23], so the problem is very unlikely to be NP-complete. However, despite much effort, no polynomial time algorithm has been devised for the problem. Determining the exact complexity of solving a parity game is a major open problem.

Three main classes of algorithms have been developed for solving parity games in practice. The *recursive* algorithm [28,41], which despite being one of the oldest algorithms, has been found to be quite competitive in practice [15]. *Strategy improvement* algorithms use a local search technique [39], similar to the simplex method for linear programming and policy iteration algorithms for solving Markov decision pro-

✉ Bart de Keijzer
b.dekeijzer@essex.ac.uk

John Fearnley
john.fearnley@liverpool.ac.uk

Sanjay Jain
sanjay@comp.nus.edu.sg

Sven Schewe
sven.schewe@liverpool.ac.uk

Frank Stephan
fstephan@comp.nus.edu.sg

Dominik Wojtczak
d.wojtzcak@liverpool.ac.uk

1 Department of Computer Science, University of Liverpool, Liverpool, UK

2 Department of Computer Science, School of Computing, National University of Singapore, Singapore, Singapore

3 Department of Computer Science and Electronic Engineering, University of Essex, Colchester, UK

4 Department of Mathematics, National University of Singapore, Singapore, Singapore

cesses. *Progress measure* algorithms define a measure that captures the winner of the game, and then use value iteration techniques to find it [22]. Each of these algorithms has inspired lines of further research, all of which have contributed to our understanding of parity games. Unfortunately, all of them are known to have exponential worst-case complexity.

Recently, Calude et al. [6] have provided a *quasi-polynomial* time algorithm for solving parity games that runs in time $O(n^{\lceil \log(c)+6 \rceil})$, where c denotes the number of priorities used in the game. Previously, the best known algorithm for parity games was a deterministic sub-exponential algorithm [21], which could solve parity games in $n^{O(\sqrt{n})}$ time, so this new result represents a significant advance in our understanding of parity games.

Their approach is to provide a compact witness that can be used to decide whether player *Even* wins a play. Traditionally, one must store the entire history of a play, so that when the players construct a cycle, we can easily find the largest priority on that cycle. The key observation of Calude et al. [6] is that a witness of poly-logarithmic size can be used instead. This allows them to simulate a parity game on an alternating Turing machine that uses poly-logarithmic space, which leads to a deterministic algorithm that uses quasi-polynomial time and space.

This new result has already inspired follow-up work. Jurdziński and Lazić [20] have developed an adaptation of the classical small progress measures algorithm [22] that runs in quasi-polynomial time. Their approach is to provide a succinct encoding of a small progress measure, which is very different from the succinct encoding developed by Calude et al. [6]. The key advantage of using progress measures as a base for the algorithm is that it avoids the quasi-polynomial space requirement of the algorithm of Calude et al., instead providing an algorithm that runs in quasi-polynomial time and near linear space.

Our contribution In this paper, we develop a progress measure-based algorithm for solving parity games that uses the succinct witnesses of Calude et al. [6]. These witnesses were designed to be used in a *forward* manner, which means that they are updated as we move along a play of the game. Our key contribution is to show that these witnesses can also be used in a *backward* manner, by processing the play backward from a certain point. This allows us to formulate a value iteration algorithm that uses (backward versions of) the witnesses of Calude et al. [6] directly.

The outcome of this is to provide a second algorithm for parity games that runs in quasi-polynomial time and near linear space. We provide a comprehensive complexity analysis of this algorithm, which is more detailed than the one given by Calude et al. [6] for the original algorithm. In particular, we show that our algorithm provides

1. a quasi-bi-linear running time for a fixed number of colours, $O(mn \log(n)^{c-1})$;
2. a quasi-bi-linear FPT bound, e.g. $O(mn\mathfrak{a}(n)^{\log\log n})$, where any other quasi-constant function can be used to replace the inverse Ackermann function $\mathfrak{a}$; and
3. an improved upper bound for a high number of colours, $O(m \cdot h \cdot n^{c_{1.45}+\log_2(h)})$

for parity games with m edges, n vertices, and c colours, where $h = \lceil 1 + c/\log(n) \rceil$ and the constant $c_{1.45} = \log_2 \mathsf{e} < 1.45$. We also provide an argument that parity games with $O(\log n)$ colours can be solved in polynomial time.

The complexity bounds (1) of our algorithm only match the bounds for the algorithm of Jurdziński and Lazić [20], while (2) and (3) are new. Moreover, we believe that it is interesting that the witnesses of Calude et al. [6] can be used in this way. The history of research into parity games has shown that ideas from the varying algorithms for parity games can often spur on further research. Our result and the work of Jurdziński and Lazić show that there are two very different ways of succinctly encoding the information that is needed to decide the winner in a parity game, and that both of them can be applied in value iteration algorithms. Moreover, implementing our progress measure is easier, as standard representations of the colours can be used. We have implemented our algorithm, and we provide some experimental results in the last section.

We introduce additionally a natural index, the au-*index*, which serves as a measure of complexity for solving a parity game using our progress measure. We present a variation of our main algorithm that runs in polynomial time when the au-index is taken as a fixed parameter of the instance. Moreover, we compare the au-index with the recently introduced *register index* introduced by Lehtinen [26]. The latter is another complexity index for parity games. Through this index, an alternative quasi-polynomial time algorithm for solving parity games was constructed by the author. We show that our au-index always exceeds the register index, which yields an alternative proof for the fact that the register index is at most $\log n + 1$, as shown previously in [26] through an entirely different argument.

Finally, we present a lower bound for our algorithm, and for the algorithm of Calude et al. [6]. We derive a family of examples upon which both of the algorithms achieve their worst-case—quasi-polynomial—running time. These are simple single player games.

Since the present paper has appeared in a preliminary form as [12], we briefly outline the contribution of this paper with respect to the preliminary conference version: The paper has been extended with a detailed analysis (and a correction) on how to compute an update of a witness in both the basic update game and the antagonistic update game. Section 9 is also entirely new, which considers the length of the witness

that is required for determining the winner of a parity game as a measure of complexity for solving said parity game. This complexity measure is compared in particular with the register index, and an alternative proof is given for why the register index is at most $\lfloor \log n \rfloor + 1$. Lastly, additional experiments have been done, which are reported on in Sect. 11, where the algorithm described in the proof of Theorem 7 (in the new Sect. 9) is evaluated.

2 Preliminaries

$\mathbb{N}$ denotes the set of positive natural numbers $\{1, 2, 3, \ldots\}$. Parity games are turn-based zero-sum games played between two players—Even and Odd, or maximiser and minimiser—over finite graphs. A parity game $\mathcal{P}$ is a tuple (V_e, V_o, E, C, ϕ), where $(V = V_e \cup V_o, E)$ is a finite directed graph with the set of vertices V partitioned into a set V_e of vertices controlled by player *Even* and a set V_o of vertices controlled by player *Odd*, $E \subseteq V \times V$ is the set of edges, $C = \{1, \ldots, c\}$ is the set of the first c natural numbers for some $c \in \mathbb{N}$, which we refer to as the *colours*, and $\phi : V \to C$ is the colour mapping. We require that every vertex has at least one outgoing edge.

A parity game $\mathcal{P}$ is played between the two players, *Even* and *Odd*, by moving a token along the edges of the graph. A play of such a game starts by placing a token on some initial vertex $v_0 \in V$. The player controlling this vertex then chooses a successor vertex v_1 such that $(v_0, v_1) \in E$ and the token is moved to this successor vertex. In the next turn, the player controlling the vertex v_1 chooses the successor vertex v_2 with $(v_1, v_2) \in E$ and the token is moved accordingly. Both players move the token over the arena in this manner and thus form a play of the game. Formally, a play of a game $\mathcal{P}$ is an infinite sequence of vertices $\langle v_0, v_1, \ldots \rangle \in V^\omega$ such that, for all $i \geq 0$, we have that $(v_i, v_{i+1}) \in E$. We write $\mathsf{Plays}_\mathcal{P}(v)$ for the set of plays of the game $\mathcal{P}$ that start from a vertex $v \in V$ and $\mathsf{Plays}_\mathcal{P}$ for the set of plays of the game. We omit the subscript when the arena is clear from the context. We extend the colour mapping $\phi : V \to C$ from vertices to plays by defining the mapping $\phi : \mathsf{Plays} \to C^\omega$ as $\langle v_0, v_1, \ldots \rangle \mapsto \langle \phi(v_0), \phi(v_1), \ldots \rangle$.

A play $\langle v_0, v_1, \ldots \rangle$ is won by player *Even* if $\limsup_{i \to \infty} \phi(v_i)$ is even, by player *Odd* if $\limsup_{i \to \infty} \phi(v_i)$ is odd.

A strategy for player *Even* is a function $\sigma : V^* V_e \to V$ such that $\big(v, \sigma(\rho, v)\big) \in E$ for all $\rho \in V^*$ and $v \in V_e$. A strategy σ is called memoryless if σ only depends on the last state ($\sigma(\rho, v) = \sigma(\rho', v)$ for all $\rho, \rho' \in V^*$ and $v \in V_e$). A play $\langle v_0, v_1, \ldots \rangle$ is consistent with σ if, for every initial sequence $\rho_n = v_0, v_1, \ldots, v_n$ of the play that ends in a state of player *Even* ($v_n \in V_e$), $\sigma(\rho_n) = v_{n+1}$ holds.

It is well known that the following conditions are equivalent: Player *Even* wins the game starting at v_0 if she has a strategy σ that satisfies that

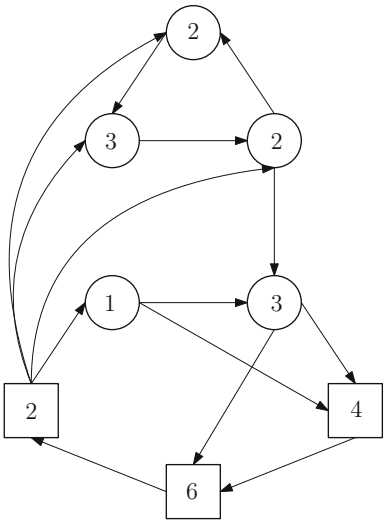

Fig. 1 An example parity game with 8 vertices, where the label of a vertex indicates its colour. The shapes of the vertices denote which player controls it: squares for player *Even*, and circles for player *Odd*. In this parity game, the three top vertices form a directed triangle in which player *Odd* wins the parity game, as player *Odd* can ensure that colour 3 is the highest colour occurring infinitely often, by always moving the token to a vertex within the triangle. On the other hand, for the remaining five vertices we observe that the subgraph induced by these vertices contains three cycles, and 6 is the highest colour in each of these cycles. Therefore, since player *Even* can make sure that the token does not escape these five vertices, player *Even* can guarantee a win if the token is located in either of these vertices

1. all plays $\langle v_0, v_1, \ldots \rangle$ consistent with σ satisfy $\limsup_{i \to \infty} \phi(v_i)$ (i.e. the highest colour that occurs infinitely often in the play) is even;
2. all plays $\langle v_0, v_1, \ldots \rangle$ consistent with σ contain a winning loop $v_i, v_{i+1}, \ldots, v_{i+k}$, that satisfies $v_i = v_{i+k}$ and $\phi(v_i) \geq \phi(v_{i+j})$ for all natural numbers $j \leq k$, where $\phi(v_i)$ is even;
3. as (1), and σ must be memoryless; or
4. as (2), and σ must be memoryless.

We use different criteria in the technical part, choosing the one that is most convenient.

An example of a parity game is depicted in Fig. 1, which depicts a game arena where the vertices are labelled by their colours, and the controlling player of a vertex is indicated by its shape: squares for player *Even*, and circles for player *Odd*.

3 QP algorithms

We discuss a variation of the algorithm of Calude et al. [6].

In a nutshell, the algorithm keeps a data structure, the witnesses, that encodes the existence of sequences of "good" events. This intuitively qualifies witnesses as a measure of

progress in the construction of a winning cycle. This intuition does not fully hold, as winning cycles are not normally identified immediately, but it gives a good intuition of the guarantees the data structure provides.

In [6], witnesses are used to track information in an alternating machine. As they are quite succinct (they have only logarithmically many entries in the number of vertices of the game, and each entry only requires logarithmic space in the number of colours), this entails the quasi-polynomial complexity.

We have made this data structure accessible for value iteration, using it in a similar way as classical progress measures. This requires a—simple—argument that witnesses can be used in a backward analysis of a run just as well as in a forward analysis. This, in turn, requires a twist in the updating rule that allows for rational decisions. For this, we equip the data structure with an order and show that the same game is still won by the same player if the antagonist can increase the value in every step.

i-Witnesses Let $\rho = v_1, v_2, \ldots, v_m$ be a prefix of a play of the parity game. An *i-witness* is a sequence of (not necessarily consecutive) positions of ρ

$$p_1, p_2, p_3, \ldots, p_{2^i},$$

of length exactly 2^i, that satisfies the following properties:

- *Position:* Each p_j specifies a position in the play ρ, so each p_j is an integer that satisfies $1 \leq p_j \leq m$.
- *Order:* The positions are ordered. So we have $p_j < p_{j+1}$ for all $j < 2^i$.
- *Evenness:* All positions other than the final one are even. Formally, for all $j < 2^i$ the colour $\phi(v_{p_j})$ of the vertex in position p_j is even.
- *Inner domination:* The colour of every vertex between p_j and p_{j+1} is dominated by the colour of p_j, or the colour of p_{j+1}. Formally, for all $j < 2^i$, the largest colour of any vertex in the subsequence $v_{p_j}, v_{p_j+1}, \ldots, v_{p_{j+1}}$ is less than or equal to $\max\{\phi(v_{p_j}), \phi(v_{p_{j+1}})\}$.
- *Outer domination:* The colour of p_{2^i} is greater than or equal to the colour of every vertex that appears after p_{2^i} in ρ. Formally, for all k in the range $p_{2^i} < k \leq m$, we have that $\phi(v_k) \leq \phi(v_{p_{2^i}})$.

Witnesses We define $C_\_ = C \cup \{\_\}$ to be the set of colours augmented with the _ symbol. A *witness* is a sequence

$$b_k, b_{k-1}, \ldots, b_1, b_0,$$

of length $k+1$—we will later see that $k = \lfloor \log_2(e) \rfloor$ is big enough, where e is the number of vertices with an even colour—where each element $b_i \in C_\_$, and that satisfies the following properties.

- *Witnessing* There exists a family of i-witnesses, one for each element b_i with $b_i \neq \_$. We refer to such an i-witness in the run ρ. We will refer to this witness as

$$p_{i,1}, \; p_{i,2}, \; \ldots, \; p_{i,2^i}.$$

- *Dominating colour* For each $b_i \neq \_$, we have that $b_i = \phi(v_{p_{i,2^i}})$. In other words, b_i is the outer domination colour of the i-witness.
- *Ordered sequences* The i-witness associated with b_i starts after j-witness associated with b_j whenever $i < j$. Formally, for all i and j with $i < j$, if $b_i \neq \_$ and $b_j \neq \_$, then $p_{j,2^j} < p_{i,1}$.

It should be noted that the i-witnesses associated with each position b_i are not stored in the witness, but in order for a sequence to be a witness, the corresponding i-witnesses must exist.

Observe that the dominating colour property combined with the ordered sequences property imply that the colours in a witness are monotonically increasing (as a function of the index), since each colour b_j (weakly) dominates all colours that appear afterwards in ρ.

Forward and backward witnesses So far, we have described *forward* witnesses, which were introduced in [6]. In this paper, we introduce the concept of *backward* witnesses, and an ordering over these witnesses, which will be used in our main result. For each play $\rho = v_1, v_2, \ldots, v_m$, we define the reverse play $\overleftarrow{\rho} = v_m, v_{m-1}, \ldots, v_1$. A backward witness is a witness for $\overleftarrow{\rho}$, or for an initial sequence of it.

Order on witnesses We first introduce an order $\succeq$ over the set $C_\_$ that captures the following requirements: even numbers are better than odd numbers, and all numbers are better than _. Among the even numbers, higher numbers are better than smaller ones, while among the odd numbers, smaller numbers are better than higher numbers. Formally, $b \succeq c$ if and only if one of the following holds:

- $c = \_$;
- b, c are both odd, and $b \leq c$;
- b, c are both even, and $b \geq c$;
- b is even and c is odd.

Then, we define an order $\sqsupseteq$ over witnesses. This order compares two witnesses lexicographically, starting from b_k and working downwards, and for each individual position the entries are compared using $\succeq$. We also define a special witness won which is $\sqsupseteq$ than any other witness.

The value of a witness An *even chain* of length m is a sequence of positions $p_1 < p_2 < p_3 < \ldots < p_m$ (with $0 \leq p_0$ and $p_m \leq n$) in ρ that has the following properties:

- for all $j \leq m$, we have that $\phi(v_{p_j})$ is even, and
- for all $j < m$ the colours in the subsequence defined by p_j and p_{j+1} are less than or equal to $\phi(p_j)$ or $\phi(p_{j+1})$. More formally, we have that all colours $\phi(v_{p_j}), \phi(v_{(p_j)+1}), \ldots, \phi(v_{p_{(j+1)}})$ are less than or equal to $\max\{\phi(v_{p_j}), \phi(v_{p_{j+1}})\}$.

For each witness $\mathbf{b} = b_k, b_{k-1}, \ldots, b_0$, we define the function $\mathsf{even}(\mathbf{b}, i) = 1$ if $b_i \neq \_$ and b_i is even. Then we define the value of the witness $\mathbf{b}$ to be $\mathsf{value}(\mathbf{b}) = \sum_{i=0}^{k} 2^i \cdot \mathsf{even}(\mathbf{b}, i)$. We can show that the value $\mathbf{b}$ corresponds to the length of an even chain in ρ that is witnessed by $\mathbf{b}$.

Lemma 1 *If* $\mathbf{b}$ *is a (forward or backward) witness of* ρ*, then there is an even chain of length* $\mathsf{value}(\mathbf{b})$ *in* ρ*.*

Proof Let i be an index such that $\mathsf{even}(\mathbf{b}, i) = 1$. By definition, the i-witness $p_{i,1}, p_{i,2}, \ldots, p_{i,2^i}$ is an even chain of length 2^i in ρ. This holds irrespective of whether $\mathbf{b}$ is a forward or backward witness.

Then, given an index $j < i$ such that $\mathsf{even}(\mathbf{b}, j) = 1$, observe that the outer domination property ensures that $\phi(p_{i,2^i}) \geq \phi(v_l)$ for all l in the range $p_{i,2^i} \leq l \leq p_{j,1}$. So, when we concatenate the i-witness with the j-witness we still obtain an even chain. Thus, ρ must contain an even chain of length $\mathsf{value}(\mathbf{b})$. □

Let $e = |\{v \in V \ : \ \phi(v) \text{ is even }\}|$ be the number of vertices with even colours in the game. Observe that, if we have an even chain whose length is strictly greater than e, then ρ must contain a cycle, since there must be at least one vertex with even colour that has been visited at least twice. Moreover, the largest priority on this cycle must be even, so this is a winning cycle for player *Even*. Thus, for player *Even* to win the parity game, it is sufficient for him to force a play that has a witness whose value is strictly greater than e.

Lemma 2 *If, from an initial state* v_0*, player* Even *can force the game to run through a sequence* ρ*, such that* ρ *has a (forward or backward) witness* $\mathbf{b}$ *with* $\mathsf{value}(\mathbf{b})$ *greater than the number of vertices with even colour, then player* Even *wins the parity game starting at* v_0*.*

3.1 Updating forward witnesses

We now show how forward witnesses can be constructed incrementally by processing the play one vertex at a time. Throughout this subsection, we will suppose that we have a play $\rho = v_0, v_1, \ldots, v_m$, and a new vertex v_{m+1} that we would like to append to ρ to create ρ'. We will use $c = \phi(v_{m+1})$ to denote the colour of this new vertex. We will suppose that $\mathbf{b} = b_k, b_{k-1}, \ldots, b_1, b_0$ is a witness for ρ, and we will construct a witness $\mathbf{d} = d_k, d_{k-1}, \ldots, d_1, d_0$ for ρ'.

We present three lemmas that allow us to perform this task.

Lemma 3 *Suppose that there exists an index* j *such that* b_i *is even for all* $i < j$*, and that* $b_i \geq c$ *or* $b_i = \_$ *for all* $i > j$*. If we set* $d_i = b_i$ *for all* $i > j$*,* $d_j = c$*, and* $d_i = \_$ *for all* $i < j$*, then* $\mathbf{d}$ *is a witness for* ρ'*.*

Proof For the indices $i > j$, observe that since $b_i \geq c$, the outer domination of the corresponding i-witnesses continues to hold. For the indices $i < j$, since we set $d_i = \_$ there are no conditions that need to be satisfied.

To complete the proof, we must argue that there is a j-witness that corresponds to d_j. This witness is obtained by concatenating the i-witnesses corresponding to the numbers b_i for $i < j$, and then adding the vertex v_{m+1} as the final position. This produces a sequence of length $1 + \sum_{i=0}^{j-1} 2^i = 2^j$ as required. Since all b_i with $i < j$ were even, the evenness condition is satisfied. For inner domination, observe that the outer domination of each i-witness ensures that the gaps between the concatenated sequences are inner dominated, and the fact that b_0 dominates sequence $v_{p_{0,1}}, \ldots, v_m$ ensures that the final subsequence is also dominated by b_0 or c. Outer domination is trivial, since v_{m+1} is the last vertex in ρ'. So, we have constructed a j-witness for ρ', and we have shown that $\mathbf{d}$ is a witness for ρ'. □

Note that, differently from Calude et al. [6], we also allow this operation to be performed in the case where c is odd.

Lemma 4 *Suppose that there exists an index* j *such that* $b_j \neq \_$*,* $c > b_j$*, and, for all* $i > j$*, either* $b_i = \_$ *or* $b_i \geq c$ *hold. Then setting* $d_i = b_i$ *for all* $i > j$*, setting* $d_j = c$*, and setting* $d_i = \_$ *for all* $i < j$ *yields a witness for* ρ'*.*

Proof For all $i > j$, we set $d_i = b_i$. Observe that this is valid, since $b_i \geq c$, and so the outer domination property continues to hold for the i-witness associated with b_i. For all $i < j$, we set $d_i = \_$, and this is trivially valid, since this imposes no requirements upon ρ'.

To complete the proof, we must argue that setting $d_j = c$ is valid. Observe that in ρ, the j-witness associated with b_j ends at a certain position $p = p_{j,2^j}$. We can create a new j-witness for ρ' by instead setting $p_{j,2^j} = m + 1$, that is, we change the last position of the j-witness to point to the newly added vertex. Note that inner domination continues to hold, since $c > b_j = \phi(v_p)$ and since v_p outer dominated ρ. All other properties trivially hold, and so $\mathbf{d}$ is a witness for ρ'. □

Lemma 5 *Suppose that for all* $j \leq k$ *either* $b_j = \_$ *or* $b_j \geq c$*. If we set* $d_i = b_i$ *for all* $i \leq k$*, then* $\mathbf{d}$ *is a witness for* ρ'*.*

Proof Since $c \leq b_j$ for all j, the outer domination of every i-witness implied by $\mathbf{b}$ is not changed. Moreover, no other property of a witness is changed by the inclusion of v_{m+1}, so by setting $\mathbf{d} = \mathbf{b}$ we obtain a witness for ρ'. □

When we want to update a witness upon scanning another state v_{m+1}, we find the largest witness that (according to $\sqsubseteq$) can be obtained by applying Lemmas 3–5.

For a given witness $\mathbf{b}$ and a vertex v_{m+1}, we denote by

- $\mathsf{ru}(\mathbf{b}, v_{m+1})$ the raw update of the witness to $\mathbf{d}$, as obtained by the update rules described above. For convenience, we alternatively write this as $\mathsf{ru}(\mathbf{b}, c)$ where $c = \phi(v_{m+1})$.
- $\mathsf{up}(\mathbf{b}, v_{m+1})$ is either $\mathsf{ru}(\mathbf{b}, v_{m+1})$ if $\mathsf{value}\big(\mathsf{ru}(\mathbf{b}, v_{m+1})\big) \leq e$ (where e is the number of vertices with even colour), or $\mathsf{up}(\mathbf{b}, v_{m+1}) = \mathsf{won}$ otherwise.

In [6], the above three update rules are used with the only difference being that the rule corresponding to Lemma 3 is only applied when c is even. In our version of the update rules, this operation can also be applied when c is odd. As the raw update is defined by choosing among all update rules the one that yields the $\sqsubseteq$-largest witness, we can prove that the update rule of Lemma 5, which leaves the witness unchanged, never needs to be applied to obtain $\mathsf{ru}(\mathbf{b}, c)$ from $\mathbf{b}$.

Lemma 6 *The witness* $\mathsf{ru}(\mathbf{b}, c)$ *is obtainable from* $\mathbf{b}$ *by applying the update rule corresponding to Lemmas 3 or 4.*

Proof It suffices to prove that when the preconditions of Lemma 5 apply, then applying the update rule of Lemma 3 instead of Lemma 5 yields a $\sqsubseteq$-higher (or equal) witness. (Note that the other update rule, i.e. the one corresponding to Lemma 4, is not applicable when the preconditions of Lemma 5 hold.)

To that end, assume that for all $j \leq k$ either $b_j = \_$ or $b_j \geq c$. We show that $\mathbf{b} \sqsubseteq \mathbf{b}'$, where $\mathbf{b}'$ is the witness resulting from applying the update rule of Lemma 3.

- Suppose first that $b_0 = \_$ then $\mathbf{b}' = b_k, \ldots, b_1, c$. Since $c \succ \_$ it holds that $\mathbf{b}' \sqsupset \mathbf{b}$.
- Suppose next that b_0 is odd. Then again $\mathbf{b}' = b_k, \ldots, b_1, c$. The colour b_0 is odd and c is either even or an odd number at most b_0, so $b_0 \preceq c$ and therefore $\mathbf{b}' \sqsupseteq \mathbf{b}$.
- Finally, suppose that b_0 is even. Let j be the least index containing a non-even value (i.e. an odd value or _). Then $\mathbf{b}' = b_k, \ldots, b_{j+1}, c, \_, \ldots$. Since b_j is _ or odd and c is either even or an odd number at most b_j, we have $b'_j = c \succ b_j$ and therefore $\mathbf{b}' \sqsupset \mathbf{b}$. □

Updating a witness $\mathbf{b}$ through applying the update rule of Lemmas 3 or 4 always results in a witness of the form $b_k, \ldots, b_{j+1}, c, \_, \ldots$ for some index j. We refer to j as the *index where the update rule is applied*.

Let us analyse more precisely at what index we need to apply the update rules of Lemmas 3 and 4 in order to obtain $\mathsf{ru}(\mathbf{b}, c)$ from $\mathbf{b}$. First observe that Lemma 4 can only be applied at a single index, namely, the unique index j that satisfies $b_j < c$ and $c \leq \min(\{b_k, \ldots, b_{j+1}\} \setminus \{\_\})$.

For the update rule of Lemma 3, there might be multiple indices in the witness where the rule can be applied. For example, when $\mathbf{b} = \_, \_, 11, 8, 8, 4, 2$ and $c = 6$, the update rule of Lemma 3 can be applied at indices 1 to 4. The next lemma shows that in case Lemma 3 can be applied on a witness, it is always applied at the leftmost index j' where it can be applied, unless $b_{j'} = c$ in which case Lemma 4 is applied at index $j' - 1$.

Lemma 7 *Suppose the update rule of Lemma 3 is applicable on* $\mathbf{b}$ *at least two indices* j' *and* j''*, where* j' *is the highest index at which the update rule is applicable. Let* $\mathbf{b}'$ *be the witness resulting from applying the update rule at* j' *and let* $\mathbf{b}''$ *be the witness resulting from applying the update rule at* j''*. Then* $b_{j'} \geq c$ *and moreover*

- *if* $b_{j'} \neq c$*, then* $\mathbf{b}' \sqsupseteq \mathbf{b}''$*;*
- *if* $b_{j'} = c$ *then* $j'' = j' - 1$*, the update rule of Lemma 4 can be applied at index* j''*, and* $\mathbf{b}'' \sqsupseteq \mathbf{b}'$*.*

Proof By definition of the update rule of Lemma 3, it holds that $b_{j'}$ is odd or _. Also, the witnesses $\mathbf{b}'$ and $\mathbf{b}''$ coincide at indices k to $j' + 1$. Moreover, at index j' we have $b'_{j'} = c$ and $b''_{j'} = b_{j'}$.

First let us assume $b_{j'} \neq c$. Then in order to determine whether $\mathbf{b}'$ or $\mathbf{b}''$ is higher in the $\sqsubseteq$-order, we need to compare their values at index j' with respect to the $\preceq$-relation.

Given that $\mathbf{b}''$ is a valid witness, we obtain the inequality $c = b''_{j''} \leq b''_{j'} = b_{j'}$. We now distinguish two cases.

- If c is even then clearly $c \succ b_{j'}$ so that $\mathbf{b}' \sqsupset \mathbf{b}''$.
- If c is odd (and by assumption not equal to $d_{j'}$), then the fact that $c \leq b_{j'}$ implies $c < b_{j'}$ and therefore $c \succ b_{j'}$, so $\mathbf{b}' \sqsupset \mathbf{b}''$.

In case $c = b_{j'}$, then all the even numbers $b_{j'-1}, \ldots, b_{j''}$ must be at least c (because $\mathbf{b}''$ is a valid witness), and since c is odd we infer that $j'' = j' - 1$ where $b_{j''} < c$. This means that the update rule corresponding to Lemma 4 is applicable at index j'' and $\mathbf{b}'' \sqsupset \mathbf{b}'$. □

The above insights lead to the following corollary, which characterises which of the two update rules are used in which cases.

Corollary 1 *The witness* $\mathsf{ru}(\mathbf{b}, c)$ *is obtained from* $\mathbf{b}$ *as follows: Let* j *be the maximum index where the update rule of Lemma 3 is applicable and let* j' *be the index at which the update rule of Lemma 4 is applicable (where we define* $j' = -1$ *if Lemma 4 is not applicable).*

- *If* $j' = j - 1 \geq 0$ *and* $b_j = c$*, apply the update rule of Lemma 4 at index* j'*.*

– *Else, if $j > j'$, apply the update rule of Lemma 3 at index j.*
– *Otherwise, apply the update rule of Lemma 4 (at index j').*

4 Basic update game

With these update rules, we define a forward and a backward basic update game. The game is played between player *Even* and player *Odd*. In this game, player *Even* and player *Odd* produce a play of the game as usual: if the token is on a position of player *Even*, then player *Even* selects a successor, and if the pebble is on a position of player *odd*, then player *Odd* selects a successor.

Player *Even* can stop any time she likes and evaluate the game using $\mathbf{b}_0 = \_, \ldots, \_$ as a starting point and the update rule $\mathbf{b}_{i+1} = \mathsf{up}(\mathbf{b}_i, v_i)$. For a forward game, she would process the partial play $\rho^+ = v_0, v_1, v_2, \ldots, v_n$ from left to right, and for the backward game she would process the partial play $\rho^- = v_n, v_{n-1}, \ldots, v_0$. In both cases, she has won if $\mathbf{b}_{n+1} = \mathsf{won}$.

Theorem 1 *If player* Even *has a strategy to win the (forward or backward) basic update game, then she has a strategy to win the parity game.*

Proof By definition, we can only have $\mathbf{b}_{n+1} = \mathsf{won}$ if at some point we created a witness whose value was more than the total number of even colours in the game. As we have argued, such a witness implies that a cycle has been created, and that the largest priority on the cycle is even. Since player *Even* can achieve this no matter what player *Odd* does, this implies that player *Even* has a winning strategy for the parity game. □

5 The data structure for the progress measure

Recall that there are two obstacles in implementing the algorithm of Calude et al. [6] as a value iteration algorithm. The first (and minor) obstacle is that it uses forward witnesses, while value iteration naturally uses backward witnesses. We have already addressed this point by introducing the same measure for a backward analysis.

The second obstacle is the lack of an order over witnesses that is compatible with value iteration. While we have introduced an order in the previous sections, this order is not a natural order. In particular, it is not preserved under update, nor does it agree with the order over values. As a simple example, consider the following two sequences:

– $\mathbf{b} = \_, 4, 2,$ and
– $\mathbf{d} = 9, 8, \_$.

While $\mathsf{value}(\mathbf{b}) = 3 > \mathsf{value}(\mathbf{d}) = 2$, $\mathbf{d} \sqsupset \mathbf{b}$. In particular, $d_2 \succ b_2$ and $d_1 \succ b_1$ hold. Yet, when using the update rules when traversing a state with colour 6, $\mathbf{b}$ is updated to $\mathbf{b}' = 6, \_, \_,$, while $\mathbf{d}$ is updated to $\mathbf{d}' = 9, 8, 6$. While $\mathbf{d} \sqsupset \mathbf{b}$ held prior to the update, $\mathbf{d}' \sqsubset \mathbf{b}'$ holds after the update. Value iteration, however, needs a natural order that will allow us to choose the successor with the higher value.

We overcome this problem by allowing the antagonist in our game, player *odd*, an extra move: prior to executing the update rule for a value $\mathbf{b}$, player *Odd* may increase the witness $\mathbf{b}$ in the $\sqsubseteq$ ordering. The corresponding *antagonistic update* is defined as follows.

$$\mathsf{au}(\mathbf{b}, v) = \min_{\sqsubseteq}\{\mathsf{up}(\mathbf{d}, v) \mid \mathbf{d} \sqsupseteq \mathbf{b}\}.$$

Observe that if $\mathbf{b} \sqsubseteq \mathbf{b}'$ then $\mathsf{au}(\mathbf{b}, v) \sqsubseteq \mathsf{au}(\mathbf{b}', v)$ because the minimum taken in $\mathsf{au}(\mathbf{b}', v)$ is over a smaller set than the minimum taken in $\mathsf{au}(\mathbf{b}, v)$. For convenience, we may alternatively write $\mathsf{au}(\mathbf{b}, c)$ to denote $\mathsf{au}(\mathbf{b}, v)$, where $c = \phi(v)$.

We will from now on refer to the raw update rules of Sect. 3.1 as follows.

Definition 1 We name the update rules corresponding to Lemmas 3 and 4 as follows.

– We use the term *update rule 1* for the update rule corresponding to Lemma 3;
– We use the term *update rule 2* for the update rule corresponding to Lemma 4;
– If $\mathsf{ru}(\mathbf{b}, c)$ can be obtained from $\mathbf{b}$ both by an application of update rule 1 and 2 (i.e. applying either update rule yields the same result), then we will regard this as an application of update rule 2.

For the remainder of this section, let $\mathbf{b}$ be a witness, v be a vertex, and $c = \phi(v)$. We will establish a way to conveniently compute $\mathsf{au}(\mathbf{b}, c)$ from $\mathbf{b}$ and c. Note that by Corollary 1, Definition 1 implies the following, which we will use in the proofs below:

Corollary 2 *Update rule 1 is solely applied on an index that contains _ or an odd value exceeding c. Hence applying rule 1 to a witness results in a witness that is $\sqsubseteq$-higher.*

We will first study in which cases it holds that $\mathsf{au}(\mathbf{b}, v) \neq \mathsf{up}(\mathbf{b}, v)$. For this to hold, it must be that the "antagonist" can choose some value $\mathbf{d}$ such that $\mathsf{up}(\mathbf{d}, v) \sqsubset \mathsf{up}(\mathbf{b}, v)$. The following lemma is straightforward and shows that we can resort to studying the raw update ru rather than up.

Lemma 8 *It holds that* $\mathsf{au}(\mathbf{b}, c) \sqsubset \mathsf{up}(\mathbf{b}, c)$ *if and only if there exists a witness* $\mathbf{d}$ *such that* $\mathbf{d} \sqsupset \mathbf{b}$ *and* $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$.

Proof $\Leftarrow$. Assume there exists a witness $\mathbf{d}$ such that $\mathbf{d} \sqsupset \mathbf{b}$ and $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$. Then $\mathsf{au}(\mathbf{b}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c) \sqsubseteq \mathsf{up}(\mathbf{b}, c)$.

$\Rightarrow$. Assume $\mathsf{au}(\mathbf{b}, c) \sqsubset \mathsf{up}(\mathbf{b}, c)$. Let $\mathbf{d}$ be the witness such that $\mathbf{d} \sqsupset \mathbf{b}$, and $\mathsf{au}(\mathbf{b}, c) = \mathsf{up}(\mathbf{d}, c)$. We distinguish two cases.

- If $\mathsf{up}(\mathbf{d}, c) \neq \mathsf{won}$ and $\mathsf{up}(\mathbf{b}, c) \neq \mathsf{won}$ we derive $\mathsf{ru}(\mathbf{b}, c) = \mathsf{up}(\mathbf{b}, c) \sqsupset \mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}, c)$.
- If $\mathsf{up}(\mathbf{d}, c) \neq \mathsf{won}$ and $\mathsf{up}(\mathbf{b}, c) = \mathsf{won}$ then $\mathsf{value}(\mathsf{ru}(b, c)) > e, \mathsf{up}(\mathbf{d}, c) = \mathsf{ru}(\mathbf{d}, c)$ and $\mathsf{value}(\mathsf{ru}(\mathbf{d}, c)) \leq e$, where e is the number of vertices with an even colour. Our choice of $k = \lfloor \log(e) \rfloor$ and the definition of the $\sqsubseteq$-relation the bounds on $\mathsf{value}(\mathsf{ru}(\mathbf{d}, c))$ and $\mathsf{value}(\mathsf{ru}(\mathbf{b}, c))$ imply that $\mathsf{ru}(\mathbf{d}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$.

This finishes the proof. □

The next lemma shows that if $\mathsf{au}(\mathbf{b}, v) \neq \mathsf{up}(\mathbf{b}, v)$, the choice $\mathbf{d}$ of the antagonist can be assumed to be a witness that differs from $\mathbf{b}$ at an index that exceeds 0.

Lemma 9 *Suppose that* $\mathsf{au}(\mathbf{b}, c) \sqsubset \mathsf{up}(\mathbf{b}, c)$ *and let* $\mathbf{d} \sqsupset \mathbf{b}$ *be a witness that satisfies* $\mathsf{ru}(\mathbf{d}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$. *Let* h *be the highest index such that* $d_h \succ b_h$. *Then* $h > 0$.

Proof First note that $\mathbf{d}$ exists per Lemma 8. Suppose next for contradiction that $h = 0$. Let j be index on which the update rule (i.e. update rule 1 or 2) is applied to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$, let i be index applied when updating $\mathbf{b}$. Since antagonist $h = 0$ and $\mathsf{ru}(\mathbf{d}, c) \neq \mathsf{ru}(\mathbf{b}, c)$ it holds that $i \neq j$. Because $h = 0$, $d = b_k, \ldots, b_1, d_0$ where $d_0 \succ b_0$. We distinguish two cases.

- If $i > j$, the situation is that

$$\begin{aligned}\mathsf{ru}(\mathbf{b}, c) &= b_k, \ldots, b_{i+1}, c, \_, \ldots \\ &\sqsupset b_k, \ldots, b_i, \ldots, b_{j+1}, c, \_, \ldots \\ &= \mathsf{ru}(\mathbf{d}, c).\end{aligned}$$

  If update rule 2 would have been used to obtain $\mathsf{ru}(\mathbf{b}, c)$ from $\mathbf{b}$, then $d_i = b_i < c$, so then neither rule 1 nor 2 could have been used to update $\mathbf{d}$ on any index $j < i$, which is a contradiction. This means that update rule 1 was used to obtain $\mathsf{ru}(\mathbf{b}, c)$ from b, and particularly that b_0 is even and $d_0 \succ b_0$ is even. In turn, this means that applying update rule 1 on $\mathbf{d}$ results in $\mathsf{ru}(\mathbf{b}, c)$ which is by assumption $\sqsubseteq$-larger than $\mathsf{ru}(\mathbf{d}, c)$. This is a contradiction, as $\mathsf{ru}(\mathbf{d}, c)$ is defined as the $\sqsubseteq$-largest witness that can be obtained by applying any of the two update rules.
- If $j > i$, the situation is that

$$\begin{aligned}\mathsf{ru}(\mathbf{b}, c) &= b_k, \ldots, b_j, \ldots, b_{i+1}, c, \_, \ldots \\ &\sqsupset b_k, \ldots, b_{j+1}, c, \_, \ldots = \mathsf{ru}(\mathbf{d}, c).\end{aligned}$$

  If rule 1 was applied to obtain $\mathsf{ru}(\mathbf{b}, c)$ from $\mathbf{b}$ then d_0 is even hence applying rule 1 on $\mathbf{d}$ would yield $\mathsf{ru}(\mathbf{b}, c)$ which is $\sqsubseteq$-better than $\mathsf{ru}(\mathbf{d}, c)$, a contradiction. If $d_i < c$ and rule 2 was applied to obtain $\mathsf{ru}(\mathbf{b}, c)$ from b, then applying rule 2 on $\mathbf{d}$ would yield $\mathsf{ru}(\mathbf{b}, c)$ which is $\sqsubseteq$-better than $\mathsf{ru}(\mathbf{d}, c)$, a contradiction. Lastly, if $d_i \geq c$ and rule 2 was applied to obtain $\mathsf{ru}(\mathbf{b}, c)$ from $\mathbf{b}$, then $i = 0$ and rule 1 was applied to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ to update index $j \geq 1$ so that $c \succ b_j$, by Corollary 2. The latter implies $\mathsf{ru}(\mathbf{d}, c) \sqsupset \mathsf{ru}(\mathbf{b}, c)$, a contradiction.

This finishes the proof. □

The following lemma states that if $\mathsf{au}(\mathbf{b}, v) \neq \mathsf{up}(\mathbf{b}, v)$, the choice $\mathbf{d}$ of the antagonist can be assumed to be a witness where the value at only one index h of $\mathbf{b}$ is raised (with respect to $\preceq$), and the value at indices less than h is set to _.

Lemma 10 *Suppose that* $\mathsf{au}(\mathbf{b}, c) \sqsubset \mathsf{up}(\mathbf{b}, c)$ *and let* $\mathbf{d} \sqsupset \mathbf{b}$ *be a witness that satisfies* $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}, c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$. *Then there exists an index* h *such that the witness* $\mathbf{d}' = b_k, \ldots, b_{h+1}, d_h, \_, \ldots$ *also satisfies* $\mathbf{d}' \sqsupset \mathbf{b}$ *and* $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}', c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$.

Proof First note that the existence of $\mathbf{d}$ is guaranteed by Lemma 8. We now construct $\mathbf{d}'$ from $\mathbf{d}$: Let h be the highest index such that $d_h \succ b_h$ and define $\mathbf{d}' = b_k, \ldots, b_{h+1}, d_h, \_, \ldots$, so that $\mathbf{d}'$ and $\mathbf{d}$ differ only at the indices 0 to $h - 1$. Note that $h \geq 1$ by Lemma 9 above, so that $\mathbf{d}'$ end in _. We will show that $\mathsf{ru}(\mathbf{d}', c) \sqsubseteq \mathsf{ru}(\mathbf{d}, c)$.

Suppose for contradiction that $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$. Then $\mathbf{d} \neq \mathbf{d}'$, i.e. the suffix $d_{h-1}, \ldots, d_0$ of $\mathbf{d}$ is not equal to _, ... Let j be the index at which update rule 1 or 2 is applied to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ and let i be the index at which update rule 1 or 2 is applied to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$.

If $j = i$ and $j \geq h$, then clearly $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$ which contradicts our assumption. If $j = i$ and $j < h$, then it must be that $j = i = 0$, as the values at indices 0 to $h - 1$ of $\mathbf{d}'$ are all _, so only update rule 1 on index 0 can be used on $\mathbf{d}'$. Hence, $\mathsf{ru}(\mathbf{d}', c) = d_k, \ldots, d_h, \_, \ldots, \_, c$ and $\mathsf{ru}(\mathbf{d}, c) = d_k, \ldots, d_h, \ldots, d_1, c$. If $d_{h-1}, \ldots, d_1$ are all _ then $\mathsf{ru}(\mathbf{d}, c) = \mathsf{ru}(\mathbf{d}', c)$ which contradicts our assumption. Otherwise, let x be the highest index among $h-1, \ldots, 1$ such that $d_x \neq \_$. Observe now that $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ agree on indices k to $x + 1$ and on coordinate x of $\mathsf{ru}(\mathbf{d}, c)$ and $\mathsf{ru}(\mathbf{d}', c)$ it holds that $d_x \succ d'_x = \_$, so $\mathsf{ru}(\mathbf{d}', c) \sqsubset \mathsf{ru}(\mathbf{d}, c)$, which again contradicts our assumption.

Therefore, $j \neq i$. Suppose first that $i = 0$ so that $j > i$. Then rule 1 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from d', and $\mathsf{ru}(\mathbf{d}', c)$ is of the form $b_k, \ldots, b_{h+1}, d_h, \_, \ldots, \_, c$ where all values $b_k, \ldots, b_{h+1}$ are each at least c or _ and d_h is at least c. If $j < h$, then let x be the highest index among $h - 1, \ldots, 1$

such that $d_x \neq \_$, and note that $x > 0$ by Lemma 9. Now observe that $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ agree on indices k to $x+1$, and at index x we have that $d_x \succ d'_x = \_$. Thus, $\mathsf{ru}(\mathbf{d}, c) \sqsupset ru(\mathbf{d}', c)$, which is a contradiction. If $j \geq h$ then rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$, and $d_j = b_j \neq \_$. However, this is in contradiction with our earlier conclusion that $b_k, \ldots, b_{h+1}$ are each at least c or $\_$, from which it follows that $d_j = b_j \geq c$ and hence rule 2 cannot be applied at index j on $\mathbf{d}$.

Suppose next that $i > 0$ and $i \neq j$. Then $i \geq h$, as neither update rule 1 nor 2 can be applied at indices $h-1$ to 1 on d'. Moreover, update rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from d'. If $j < i$, then $d_k \ldots d_h$ are all at least c, which means that it is impossible to apply update rule 2 to $\mathbf{d}'$ at index i, a contradiction. If $j > i$ then the update of $\mathbf{d}'$ at index i implies that $d_k, \ldots, d_{i+1}$ are all at least c, so that $\mathsf{ru}(\mathbf{d}, c)$ is obtained from $\mathbf{d}$ through update rule 1. By Corollary 2, index j of $\mathsf{ru}(\mathbf{d}, c)$ has the value $c \succ b_j$, and b_j is the value at index j of $\mathsf{ru}(\mathbf{d}', c)$, so $\mathsf{ru}(\mathbf{d}, c) \sqsupset \mathsf{ru}(\mathbf{d}', c)$, a contradiction. □

Using the above lemma, we may now provide a concrete efficient way to compute $\mathsf{au}(\mathbf{b}, c)$.

Definition 2 Let c be either a colour or $\_$. We denote by $\mathsf{succ}_{\preceq}(c)$ the immediate successor of c in the $\preceq$-ordering.

Let $\mathbf{b}$ be a witness, j be an index, and $v_j = \min(\{b_k, \ldots, b_{j+1}\} \setminus \{\_\})$, i.e. v_j is the least colour occurring in $\mathbf{b}$ at any index exceeding j. The *minimal raisable index* of $\mathbf{b}$ is the minimal index $j \geq 1$ such that $\mathsf{succ}_{\preceq}(b_j)$ exists (i.e. b_j is not the highest even colour) and $v_j \geq \mathsf{succ}_{\preceq}(b_j)$. The corresponding *minimal raisable colour* s is defined as the $\preceq$-minimal colour s such that $s \succ b_j$ and $b_k, \ldots, b_{j+1}, s, \_, \ldots$ is a valid witness. Note that if b_j is a colour, then $s = \mathsf{succ}_{\preceq}(b_j)$, and if $b_j = \_$ then $s = \max\{s' : s' \leq v_j$ and s' is an odd colour$\}$.

Let j be the minimal raisable index of $\mathbf{b}$ and let s be the corresponding minimal raisable colour. Let $\mathbf{d} = b_k, \ldots, b_{j+1}, s, \_, \ldots$. We will prove through the following lemmas that $\mathsf{au}(\mathbf{b}, \mathbf{c})$ is the $\sqsubseteq$-minimum of $\mathsf{ru}(\mathbf{d}, c)$ and $\mathsf{up}(\mathbf{b}, c)$, unless the exceptional case holds that c is odd, c is not the lowest colour, and there is a maximal index h such that $b_h = c$. In the latter case it holds that $\mathsf{au}(\mathbf{b}, c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$.

We start by proving our claim for the exceptional case.

Lemma 11 *Suppose c is odd and not the lowest colour, and suppose that there is a maximal index h such that $b_h = c$. Then $\mathsf{au}(\mathbf{b}, c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$*

Proof Let $\mathbf{d} = b_k, \ldots, b_{h+1}, \mathsf{succ}_{\preceq}(c), \_, \ldots$, and observe that under the given preconditions, $\mathsf{ru}(\mathbf{d}, c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$. Therefore $\mathsf{au}(\mathbf{b}, c) \sqsubseteq \mathsf{ru}(\mathbf{d}, c)$.

Let h' be the highest index such that $b_{h'} < c$ or $h' = 0$. Assume first that $\mathsf{up}(\mathbf{b}, c) = \mathsf{au}(\mathbf{b}, c)$. Since c is odd, $\mathsf{up}(\mathbf{b}, c) \neq \mathsf{won}$ and therefore $\mathsf{up}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{b}, c) = b_k, \ldots, b_{h+1}, c, \ldots, b_{h'+1}, c, \_, \ldots$, where we note that for the last equality to hold we take into account the first point of Lemma 1. This yields $\mathsf{ru}(\mathbf{b}, c) \sqsupset \mathsf{ru}(\mathbf{d}, c) \sqsupseteq \mathsf{au}(\mathbf{b}, c)$ and contradicts $\mathsf{up}(\mathbf{b}, c) = \mathsf{au}(\mathbf{b}, c)$.

Therefore, $\mathsf{up}(\mathbf{b}, c) \sqsupset \mathsf{au}(\mathbf{b}, c)$, so by Lemma 8 we may assume that there exists a witness $\mathbf{d}'$ such that $\mathbf{d}' \sqsupset \mathbf{b}$ and $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}', c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$ and by Lemma 10 we may assume that $\mathbf{d}' = b_k, \ldots, b_{j+1}, d'_j, \_, \ldots$ for some $j \geq 1$ where $d'_j \succ b_j$.

If $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$, the claim follows. Suppose for contradiction that $\mathsf{ru}(\mathbf{d}', c) \sqsubset \mathsf{ru}(\mathbf{d}, c)$. We distinguish three cases.

- If update rule 1 was used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ and $j \geq h$, then the highest index where $\mathsf{ru}(\mathbf{d}', c)$ differs from $\mathsf{ru}(\mathbf{d}, c)$ is j and the values of the updated witnesses at index j are d'_j and b_j, respectively. From $d'_j \succ b_j$ it follows that $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$, which is a contradiction.
- If update rule 1 was used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ and $j < h$, then $\mathsf{ru}(\mathbf{d}', c)$ coincides with $\mathsf{ru}(\mathbf{d}, c)$ at indices k to h and differs at some maximal index x in the range $h-1$ to 0. The value at index x of $\mathsf{ru}(\mathbf{d}, c)$ is the $\preceq$-minimal value $\_$, which gives us $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$, a contradiction.
- If update rule 2 was used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$, then update rule 2 was applied on $\mathbf{d}'$ at an index at most $h-1$: Otherwise c would appear in $\mathsf{ru}(\mathbf{d}', c)$ either at an index x in the range between k and h, which would mean that $d' = b_k, \ldots, b_{x+1}, d'_x, \_, \ldots$ where $d'_x \succ b_x, d'_x < c$ and b_x is an odd number exceeding c, which would mean that $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$ if $x > h$, and $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$ if $x = h$, a contradiction. Given that update rule 2 was applied on $\mathbf{d}'$ at an index at most h, the values $d'_k, \ldots, b'_h$ are all at least c. Therefore, the values of $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ coincide at indices k to h. And must differ at some maximal index x in the range $h-1$ to 0. The value at index x of $\mathsf{ru}(\mathbf{d}, c)$ is the $\preceq$-minimal value $\_$, which gives us $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$, a contradiction.

This finishes the proof. □

In the following lemma, we show that it suffices for the antagonist to consider as an antagonistic choice just the witness $\mathbf{d} = b_k, \ldots, b_{j+1}, s, \_, \ldots$ where j is the minimal raisable index of $\mathbf{b}$ and s is the corresponding minimal raisable colour.

Lemma 12 *Suppose that the preconditions of Lemma 11 do not hold. If $\mathsf{au}(\mathbf{b}, c) \neq \mathsf{up}(\mathbf{b}, c)$, then $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}, c)$, where $\mathbf{d} = b_k, \ldots, b_{j+1}, s, \_, \ldots$, and where j is the minimal raisable index of $\mathbf{b}$ and s the corresponding minimal raisable colour.*

Proof By Lemma 8, there exists a $\mathbf{d}'$ such that $\mathsf{au}(\mathbf{b}, c) = \mathsf{ru}(\mathbf{d}', c) \sqsubset \mathsf{ru}(\mathbf{b}, c)$, and by Lemma 10 we may assume that $\mathbf{d}' = b_k, \ldots, b_{h+1}, d'_h, \_, \ldots$ for some $h \geq 1$, where $d'_h \succ b_h$.

We assume that $\mathbf{d}' \neq \mathbf{d}$ and prove that $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$. This will imply that $\mathsf{ru}(\mathbf{d}, c) \sqsubseteq \mathsf{au}(\mathbf{b}, c)$, and since $\mathbf{d} \sqsupset \mathbf{b}$ it will follow that $\mathsf{ru}(\mathbf{d}, c) = \mathsf{au}(\mathbf{b}, c)$ which settles the claim. To prove that $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$, we distinguish two cases in which $\mathbf{d}'$ could differ from $\mathbf{d}$: Either $j \neq h$ or $j = h$ and $d'_j \neq d_j = s$.

First, let us assume that $j \neq h$. Then by the definition of j being the minimal raisable index, $h > j$. So the highest index where $\mathbf{d}'$ differs from $\mathbf{d}$ is h, where $d'_h \succ b_h = d_h$.

If rule 1 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ then $\mathsf{ru}(\mathbf{d}', c) = b_k, \ldots, b_{h+1}, d'_h, \_, \ldots, \_, c$. Since this means all of $b_k, \ldots, b_{h+1}$ are _ or at least c, the value of $\mathsf{ru}(\mathbf{d}, c)$ at index h is either b_h or c (where the latter may happen in case $\mathsf{ru}(\mathbf{d}, c)$ is obtained from $\mathbf{b}$ through applying rule 2, and $b_h < c$). If $\mathsf{ru}(\mathbf{d}, c)_h = b_h$ then by $b_h \prec d'_h$ it holds that $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$. If $\mathsf{ru}(\mathbf{d}, c)_h = c$ then rule 2 rather than rule 1 has been used at index h to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ (where we emphasize that we consider the case $c \neq b_h$, as if $c = b_h$, then this was already taken care of), which means that $c \succ b_h = d_h$. If d'_h is odd, then b_h is a larger odd number or _, and in both these cases rule 2 cannot be applied to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$, so d'_h is even and at least c. If $d'_h > c$ then $d'_h \succ c$ so $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$. If $d'_h = c$ then $\mathsf{ru}(\mathbf{d}', c) = b_k, \ldots, b_{h+1}, c, \_, \ldots, \_, c$ and $\mathsf{ru}(\mathbf{d}, c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$. So $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$ by comparing the values of the two witnesses at index 0.

If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ then $\mathsf{ru}(\mathbf{d}', c) = b_k, \ldots, b_{x+1}, c, \_, \ldots$ for some index $x \geq h > j$. We distinguish various cases.

- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ and $x > h$, then rule 2 is applied at the same index x on $\mathbf{d}$, and $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$, which implies our claim that $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$.
- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at index h and if $x = h$, then $\mathsf{ru}(\mathbf{d}, c) = \mathsf{ru}(\mathbf{d}', c)$ as well.
- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at an index $i > h$ and if $x = h$, then $c > d_i = b_i = d'_i$ and rule 2 must also have been applied at index i to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$, i.e. $i = x$, a contradiction.
- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at an index $i < h$ and if $x = h$ then $\mathsf{ru}(\mathbf{d}, c) = b_k, \ldots, b_h, \ldots, b_{i+1}, c, \_, \ldots$ and $\mathsf{ru}(\mathbf{d}', c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$. Because $b_h \prec d'_h$, $b_h \geq c \geq d'_h$, it follows that d_h is _ or odd and at least c. We know that $d_h \neq c$ by assumption that the preconditions of Lemma 11 do not apply, so d_h is _ or odd and greater than c. Therefore, $c \succ b_h$ and $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$.
- If rule 1 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$, then $\mathsf{ru}(\mathbf{d}, c) = b_k, \ldots, b_h, \ldots, b_{j+1}, s, \_, \ldots, \_, c$. This means that each of $b_k, \ldots, b_h, \ldots, b_{j+1}$ is either _ or at least c, and therefore $x = h$, $d'_h \succ b_h$ and $d'_h < c$, so that b_h is odd and not the least colour, and $\mathsf{ru}(\mathbf{d}', c) = b_k, \ldots, b_{h+1}, c, \_, \ldots$. If $c = b_h$ then c is an odd colour that is not the least colour and c occurs at index h in $\mathbf{b}$, a contradiction with the assumption that the preconditions of Lemma 11 do not hold. If $c < b_h$, then since d'_h is a successor of b_h satisfying $d'_h < b_h$, we know that b_h must be odd, so $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ agree on indices k to $h+1$ and $\mathsf{ru}(\mathbf{d}', c)_h = c \succ b_h = \mathsf{ru}(\mathbf{d}, c)_h$; therefore $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$.

The above establishes that $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$ if $j \neq h$. It remains to prove that also $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$ if $j = h$ and $d'_j \neq s$. Assuming that $j = h$ and $d'_j \neq s$, it holds that $d'_j \succ s$ because of the definition of s as the minimal raisable colour. We will again distinguish a lot of cases.

- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at an index exceeding j, then rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ at the same index, and $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$.
- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at index j, and $d'_j < c$, then clearly also $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{d}, c)$.
- If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$ at index j and $d'_j \geq c$, then rule 1 is used at index 0 to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$, hence the highest index where $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ may possibly differ is j, where $\mathsf{ru}(\mathbf{d}, c)_j = c$ and $\mathsf{ru}(\mathbf{d}', c)_j = d'_j$. We see that d'_j must be even, as $d'_j \succ s$ and $d'_j > s$. Therefore, $c \leq d'_j$. If $c < d'_j$, we infer $d'_j \succ c$ hence $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$. If $c = d'_j$ then the highest index where $\mathsf{ru}(\mathbf{d}', c)$ differs from $\mathsf{ru}(\mathbf{d}, c)$ is 0, and $\mathsf{ru}(\mathbf{d}', c) \sqsupseteq \mathsf{ru}(\mathbf{d}, c)$ by $c \succ \_$.
- If rule 1 is used to obtain $\mathsf{ru}(\mathbf{d}, c)$ from $\mathbf{d}$, then $s \geq c$. If rule 1 is also used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$ then $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}', c)$ agree on indices k to $j+1$ and $\mathsf{ru}(\mathbf{d}', c)_j = d'_j \succ s = \mathsf{ru}(\mathbf{d}, c)_j$, so $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$. If rule 2 is used to obtain $\mathsf{ru}(\mathbf{d}', c)$ from $\mathbf{d}'$, then it must be at index j. So then $d'_j < c \leq s$, and by $s \prec d'_j$ it must be that s is odd. Thus, if $c < s$ then $c \succ s$ and $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{d}, c)$ as $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{d}, c)$ agree at indices k to $j+1$ and $\mathsf{ru}(\mathbf{d}', c)_j = c \succ s = \mathsf{ru}(\mathbf{d}, c)_j$. Lastly, if $c = s$ then we prove that $\mathsf{ru}(\mathbf{b}, c) \sqsubseteq \mathsf{ru}(\mathbf{d}', c)$, which contradicts our assumption that $\mathsf{ru}(\mathbf{d}', c) = \mathsf{au}(\mathbf{b}, c) \neq \mathsf{up}(\mathbf{b}, c)$:
  Since $c = s$ is defined as the minimal raisable colour of $\mathbf{b}$, it holds that b_j is either _ or an odd colour greater than $s = c$. The witness $\mathsf{ru}(\mathbf{b}, c)$ is therefore obtained from $\mathbf{b}$ by applying rule 1 or 2 at an index at most j.
  This means that either $\mathsf{ru}(\mathbf{b}, c)_j = c = s \prec d'_j$, or it holds that $\mathsf{ru}(\mathbf{b}, c)_j = b_j \prec s = c \prec d'_j$ (where this case distinction holds because of the definition of s as

the minimum raisable colour). In the former case, we see that $\mathsf{ru}(\mathbf{d}', c)_j = c = \mathsf{ru}(\mathbf{b}, c)_j$ and $\mathsf{ru}(\mathbf{d}', c)$ agrees with $\mathsf{ru}(\mathbf{b}, c)$ on all other indices as well, which would mean $\mathsf{ru}(\mathbf{d}', c) = \mathsf{ru}(\mathbf{b}, c)$, a contradiction. In the latter case, $\mathsf{ru}(\mathbf{d}', c)$ and $\mathsf{ru}(\mathbf{b}, c)$ agree at indices k to $j+1$ and at index j it holds that $\mathsf{ru}(\mathbf{d}', c)_j = c \succ b_j = \mathsf{ru}(\mathbf{b}, c)_j$, so that $\mathsf{ru}(\mathbf{d}', c) \sqsupset \mathsf{ru}(\mathbf{b}, c)$.

This finishes the proof. □

Combining the lemmas above yields as a corollary an approach for efficient computation of $\mathsf{au}(\mathbf{b}, c)$.

Corollary 3 *Let j be the minimal raisable index and s be the minimal raisable colour of $\mathbf{b}$, and define $\mathbf{d} = b_k, \ldots, b_{j+1}, s, \_, \ldots$. If c is odd, not the lowest colour, and there exists a maximal index h such that $b_h = c$, then*

$$\mathsf{au}(\mathbf{b}, c) = b_k, \ldots, b_{h+1}, c, \_, \ldots.$$

Otherwise,

$$\mathsf{au}(\mathbf{b}, c) = \min_{\sqsubseteq}\{\mathsf{ru}(\mathbf{d}, c), \mathsf{up}(\mathbf{b}, c)\}.$$

6 Antagonistic update game

The antagonistic update game is played like the basic update game, but uses the antagonistic update rule, i.e. player *Even* and *odd* execute a play of the game as usual: if the token is on a position of player *Even*, then player *Even* selects a successor, and if the token is on a position of player *odd*, then player *Odd* selects a successor.

Player *Even* can stop any time she likes and evaluate the game using $\mathbf{b}_0 = \_, \ldots, \_$ as a starting point and the update rule $\mathbf{b}_{i+1} = \mathsf{au}(\mathbf{b}_i, v_i)$. For a forward game, she would process the partial play $\rho^+ = v_0, v_1, v_2, \ldots, v_n$ from left to right, and for the backward game she would process the partial play $\rho^- = v_n, v_{n-1}, \ldots, v_0$. In both cases, she has won if $\mathbf{b}_{n+1} = \mathsf{won}$.

As an example, consider a prefix 1, 6, 2, 1, 4, 6 of a play of a parity game with colours 1 to 6. Forward evaluation of the play would result in the witness (_, 6, _), by starting from (_, _, _) and following the sequence of antagonistic updates (_, _, 1), (_, _, 6), (_, 5, 2), (_, 3, 1), (_, 4, _), (_, 6, _). Backward evaluation of the play would result in the witness (_, 6, 1), by starting from (_, _, _) and following the sequence of antagonistic updates (_, _, 6), (_, 5, 4), (_, 3, 1), (_, 3, 2), (_, 6, _), (_, 6, 1). Note that from the above sequences of updates it is easy to deduce the witness of a shorter prefix only in the case of forward evaluation, e.g. (_, _, 6) is the witness associated to forward evaluation of prefix (1, 6), which simply occurs as a second element in the above sequence of antagonistic updates. However, the witness resulting from backward evaluation of prefix 1, 6 does not occur in the above sequence and has to be recomputed (which will result in witness (_, 5, 1)).

Theorem 2 *If player* Even *has a strategy to win the (forward or backward) antagonistic update game, then she has a strategy to win the parity game.*

Proof We first look at the evaluation of a play $\rho^+ = v_0, v_1, v_2, \ldots, v_n$ or $\rho^- = v_n, v_{n-1}, \ldots, v_0$ in a forward or backward game, respectively. In an antagonistic game, this will lead to a sequence $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n+1}$, while it leads to a sequence $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n+1}$ when using the basic update rule. We show by induction that $\mathbf{b}_i \sqsupseteq \mathbf{a}_i$ holds.

For an induction basis, $\mathbf{b}_0 = \mathbf{a}_0 = \_, \ldots, \_$.

For the induction step, if $\mathbf{b}_i \sqsupseteq \mathbf{a}_i$, then

$$\mathbf{a}_{i+1} = \mathsf{au}(\mathbf{a}_i, v_i) = \min_{\sqsubseteq}\left\{\mathsf{up}(\mathbf{c}, v_i) \mid \mathbf{c} \sqsupseteq \mathbf{a}_i\right\} \sqsubseteq^{IH} \mathsf{up}(\mathbf{b}_i, v_i) = \mathbf{b}_{i+1}.$$

Thus, when player *Even* wins the (forward or backward) antagonistic update game, then she wins the (forward or backward) basic update game using the same strategy. □

It remains to show that, if player *Even* has a strategy to win the parity games, then she has a strategy to win the antagonistic update game. For this, we will use the fact that she can, in this case, make sure that the highest number that occurs infinitely often on a run is even. We exploit this in two steps. We first introduce a $\downarrow_x$ operator, for every even number x, that removes all but possibly one entry with numbers smaller than x, and adjust the one that possibly remains to $x-1$. We then argue that, when there are no higher numbers than x, this value of the witnesses obtained after this operator are non-decreasing w.r.t. $\sqsupseteq$, and increase strictly with every occurrence of x.

Formally we define, for a witness $\mathbf{b} = b_k, b_{k-1}, \ldots, b_0$ and an even number x, the following.

- $\mathbf{b} \downarrow_x$ to be $\mathbf{b}$ if, for all $i \leq k$, $b_i = \_$ or $b_i \geq x$ holds.
- Otherwise, let $i = \max\{s \leq k \mid b_s \neq \_ \text{ and } b_s < x\}$. We define $\mathbf{b} \downarrow_x = b'_k, b'_{k-1}, \ldots, b'_0$ with $b'_j = b_j$ for all $j > i$, $b'_i = x - 1$, and $b'_j = \_$ for all $j < i$.

Lemma 13 *The $\downarrow_x$ operator provides the following guarantees:*

1. $\mathbf{b} \sqsupset \mathbf{a} \;\Rightarrow\; \mathbf{b}\downarrow_x \sqsupseteq \mathbf{a}\downarrow_x$
2. $\phi(v) < x \;\Rightarrow\; \mathsf{up}(\mathbf{b}, v)\downarrow_x \sqsupseteq \mathbf{b}\downarrow_x$
3. $\phi(v) < x \;\Rightarrow\; \mathsf{au}(\mathbf{b}, v)\downarrow_x \sqsupseteq \mathbf{b}\downarrow_x$
4. $\phi(v) = x \;\Rightarrow\; \mathsf{up}(\mathbf{b}, v)\downarrow_x \sqsupset \mathbf{b}\downarrow_x$
5. $\phi(v) = x \;\Rightarrow\; \mathsf{au}(\mathbf{b}, v)\downarrow_x \sqsupset \mathbf{b}\downarrow_x$

Proof For (1), let $i \leq k$ be the highest position with $b_i \neq a_i$, and thus with $b_i \succ a_i$ (as $\mathbf{b} \sqsupset \mathbf{a}$). If $b_i \succeq x$ or $x + 1 \succeq a_i$, the claim follows immediately (and we have $\mathbf{b} \downarrow_x \sqsupset \mathbf{a} \downarrow_x$). For the case $x \succ b_i \succ a_i \succ x + 1$, this position would be replaced by $x - 1$ and all smaller positions by _, through the $\downarrow_x$ operator (and we have $\mathbf{b} \downarrow_x = \mathbf{a} \downarrow_x$).

For (2), the highest position $i \leq k$ for which $\mathbf{a} = \mathsf{up}(\mathbf{b}, v)$ and $\mathbf{b}$ differ (if any) satisfies $a_i \prec x$ and $b_i \prec x$ (the latter holds because otherwise v does not overwrite position i by this update rule). If $b_i \prec x + 1$, then we get $\mathsf{up}(\mathbf{b}, v) \downarrow_x \sqsupset \mathbf{b} \downarrow_x$; otherwise we get $\mathsf{up}(\mathbf{b}, v) \downarrow_x = \mathbf{b} \downarrow_x$.

(3) follows from (1) and (2).

For (4), $\mathbf{a} = \mathsf{up}(\mathbf{b}, v)$ and $\mathbf{b}$ differ in some highest position $i \leq k$, and for that position, $x = a_i \succ b_i$ holds. Thus, $\mathsf{up}(\mathbf{b}, v) \downarrow_x \sqsupset \mathbf{b} \downarrow_x$.

(5) follows with (1) and (4). □

This almost immediately implies the correctness.

Theorem 3 *If player* Even *can win the parity game from a position v, then she can win the (forward and backward) antagonistic update game from v.*

Proof Player *Even* can play such that the highest colour that occurs in a run infinitely many times is even. She can thus in particular play to make sure that, at some point in the run, an even colour x has occurred more often than the size of the image of $\downarrow_x$ after the last occurrence of a priority higher than x. By Lemma 13, evaluating the forward or backward antagonistic update game at this point will lead to a win of player *Even*. □

These results directly provide the correctness of all four games described.

Corollary 4 *Player* Even *can win the forward and backward antagonistic and basic update game from a position v if, and only if, she can win the parity game from v.*

7 Value iteration

The antagonistic update game offers a direct connection to *value iteration*. For value iteration, we use a *progress measure*, a function $\iota : V \to \mathbb{W}$, where $\mathbb{W}$ denotes the set of possible backward witnesses. That is, a progress measure assigns a backward witness to each vertex.

Let $\mathbf{b}_v = \max_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\}$ for $v \in V_e$ and $\mathbf{b}_v = \min_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\}$ for $v \in V_o$. We say that ι can be lifted at v if $\iota(v) \sqsubset \mathbf{b}_v$. When ι is liftable at v, we define by $\mathsf{lift}(\iota, v)$ the function ι' with $\iota'(v) = \mathbf{b}_v$ and $\iota'(v') = \iota(v')$ for all $v' \neq v$. We extend the lift operation to every non-empty set $V' \subseteq V$ of liftable positions, where $\iota' = \mathsf{lift}(\iota, V')$ updates all values $v \in V'$ concurrently.

A progress measure is called consistent if it cannot be lifted at any vertex $v \in V$. The *minimal consistent progress measure* $\iota_{\min}$ is the smallest (w.r.t. the partial order in the natural lattice defined by pointwise comparison) progress measure that satisfies

- for all $v \in V_e$ that $\iota(v) \sqsupseteq \max_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\}$, and
- for all $v \in V_o$ that $\iota(v) \sqsupseteq \min_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\}$.

As an example, consider a very simple parity game with three colours and three vertices v_1, v_2, v_3, where $\phi(v_i) = i$ for all i. Vertices v_3 and v_2 are controlled by the even player and v_1 is controlled by the odd player. There are four arcs: $(v_1, v_3), (v_3, v_2), (v_2, v_1), (v_3, v_1)$. When starting with the progress measure that assigns all vertices the minimal witness (_, _). After the first lift operation, all three witnesses change, and we obtain the progress measure $\iota(v_1) = (\_, 1), \iota(v_2) = (\_, 2), \iota(v_3) = (\_, 3)$. For the second iteration operation, it is easy to verify that no further vertex can be lifted, so the minimal consistent progress measure is attained after a single lift. Indeed, the odd player (trivially) wins everywhere in this parity game.

As $\mathsf{au}(\mathbf{b}, v)$ is monotone in $\mathbf{b}$ by definition and the state space is finite, we get the following.

Lemma 14 *The minimal consistent progress measure $\iota_{\min}$ is well defined.*

Proof First, a consistent progress measure always exists: the function that maps all states to won is a consistent progress measure.

Second if we have two consistent progress measures ι and ι', then the pointwise minimum $\iota'' : v \mapsto \min_{\sqsubseteq}\{\iota(v), \iota'(v)\}$ is a consistent progress measure. To see this, we assume w.l.o.g. that $\iota(v) \sqsubseteq \iota'(v)$.

For $v \in V_e$ we get $\iota''(v) = \iota(v) \sqsupseteq \max_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\} \sqsupseteq \max_{\sqsubseteq}\{\mathsf{au}(\iota''(s), v) \mid (v, s) \in E\}$, using that $\iota''(s) \sqsubseteq \iota(s)$ holds for all $s \in V$.

Likewise, we get for $v \in V_o$ that $\iota''(v) = \iota(v) \sqsupseteq \min_{\sqsubseteq}\{\mathsf{au}(\iota(s), v) \mid (v, s) \in E\} \sqsupseteq \min_{\sqsubseteq}\{\mathsf{au}(\iota''(s), v) \mid (v, s) \in E\}$, using again that $\iota''(s) \sqsubseteq \iota(s)$ holds for all $s \in V$.

As the state space is finite, we get the minimal consistent progress measure as a pointwise minimum of all consistent progress measures. □

Moreover, we can compute the minimal consistent progress measure by starting with the initial progress measure ι_0, which maps all vertices to the minimal witness _, ..., _, and iteratively applying lifting.

Lemma 15 *The minimal consistent progress measure $\iota_{\min}$ can be obtained by any sequence of lift operations on liftable positions, starting from ι_0.*

Proof We show that, for any sequence $\iota_0, \iota_1, \ldots, \iota_n$ of progress measures constructed by a sequence of lift operations, for all $v \in V$, and for all $i \leq n$, $\iota_i(v) \sqsubseteq \iota_{\min}(v)$ holds.

For the induction basis, $\iota_0(v)$ is the minimal element for all $v \in V$, such that $\iota_0(v) \sqsubseteq \iota_{\min}(v)$ holds trivially.

For the induction step, let $V_i \subseteq V$ be a set of liftable positions for ι_i and $\iota_{i+1} = \mathsf{lift}(\iota_i, V_i)$. We now make the following case distinction.

- For $v \in V_i \cap V_e$, we have $\iota_{i+1}(v) = \max_{\sqsubseteq}\{\mathsf{au}(\iota_i(s), v) \mid (v, s) \in E\} \sqsubseteq_{IH} \max_{\sqsubseteq}\{\mathsf{au}(\iota_{\min}(s), v) \mid (v, s) \in E\} \sqsubseteq \iota_{\min}(v)$.
- For $v \in V_i \cap V_o$, we have $\iota_{i+1}(v) = \min_{\sqsubseteq}\{\mathsf{au}(\iota_i(s), v) \mid (v, s) \in E\} \sqsubseteq_{IH} \min_{\sqsubseteq}\{\mathsf{au}(\iota_{\min}(s), v) \mid (v, s) \in E\} \sqsubseteq \iota_{\min}(v)$.
- For $v \notin V_i$, we have $\iota_{i+1}(v) = \iota_i(v) \sqsubseteq_{IH} \iota_{\min}(v)$.

This closes the induction step.

While we have proven that the value of the progress measures cannot surpass the value of $\iota_{\min}$ at any vertex, each liftable progress measure ι_i is succeeded by a progress measure ι_{i+1}, which is nowhere smaller, and strictly increasing for some vertices. Thus, this sequence terminates eventually by reaching a non-liftable progress measure. But non-liftable progress measures are consistent.

Thus, we eventually reach a consistent progress measure ι_n which is pointwise no larger than $\iota_{\min}$, i.e. we eventually reach $\iota_{\min}$. □

Once it is established that $\iota_{\min}(v) = \mathsf{won}$ holds, it is straightforward to obtain a winning strategy of player *Even* in the antagonistic update game.

Lemma 16 *If $\iota_{\min}(v) = \mathsf{won}$, then player* Even *has a strategy to win the antagonistic update game when starting from v.*

Proof We can construct the strategy in the following way: starting in state $v_n = v$, where n is the length of the play we will create, player *Even* selects for a state $v_i \in V_e$ with $i > 0$ a successor v_{i-1} such that $\iota_i(v_i) \sqsubseteq \mathsf{au}(\iota_{i-1}(v_{i-1}), v_i)$. Note that such a successor must always exist. Note also that, if $v_i \in V_o$ with $i > 0$, then $\iota_i(v_i) \sqsubseteq \mathsf{au}(\iota_{i-1}(v_{i-1}), v_i)$ holds for all successors v_{i-1} of v_i by definition.

Assume that player *Even* selects a successor from her vertices as described above, and $v_n, v_{n-1}, \ldots, v_0$ is a play created this way. Let $\mathbf{b}_0 = \_, \ldots, \_$ be the minimal element of $\mathbb{W}$, and $\mathbf{b}_{i+1} = \mathsf{au}(\mathbf{b}_i, v_{i+1})$. Then we show by induction that $\mathbf{b}_i \sqsupseteq \iota_i(v_i)$.

For the induction basis, we have $\mathbf{b}_0 = \iota_0(v_0)$ by definition. For the induction step, we have $\iota_{i+1}(v_{i+1}) \sqsubseteq \mathsf{au}(\iota_i(v_i), v_{i+1}) \sqsubseteq^{IH} \mathsf{au}(\mathbf{b}_i, v_{i+1}) = \mathbf{b}_{i+1}$.

Thus, we get $\mathbf{b}_n \sqsupseteq \iota_n(v_n) = \mathsf{won}$, and player *Even* wins the antagonistic update game. □

At the same time, player *Even* cannot win from any vertex v with $\iota_{\min}(v) \neq \mathsf{won}$, and $\iota_{\min}$ provides a witness strategy for player *Odd* for this.

Lemma 17 *Player* Even *cannot win from any vertex v with $\iota_{\min}(v) \neq \mathsf{won}$, and $\iota_{\min}$ provides a witness strategy for player* Odd.

Proof We recall that the construction of $\iota_{\min}$ by Lemma 15 provides

- $\iota_{\min}(v) \sqsupseteq \max_{\sqsubseteq}\{\mathsf{au}(\iota_{\min}(s), v) \mid (v, s) \in E\}$ for $v \in V_e$, and
- $\iota_{\min}(v) \sqsupseteq \min_{\sqsubseteq}\{\mathsf{au}(\iota_{\min}(s), v) \mid (v, s) \in E\}$ for $v \in V_o$.

The latter provides the existence of some particular successor s of v with $\iota_{\min}(v) \sqsupseteq \mathsf{au}(\iota_{\min}(s), v)$. The witness strategy of player *Odd* is to always choose such a vertex.

Let $\rho = v_n, v_{n-1}, v_{n-2}, \ldots, v_1$ be a sequence obtained by any strategy of player *Even* from a starting vertex v_n with $\iota_{\min}(v_n) \neq \mathsf{won}$, such that player *Even* chooses to evaluate the backward antagonistic update game after ρ, and ρ, v_0 an extension in line with the strategy of player *Odd*.

We first observe that $\iota_{\min}(v_{i+1}) \sqsupseteq \mathsf{au}(\iota_{\min}(v_i), v_{i+1})$ holds for all $i < n$, either by the choice of the successor of v_{i+1} of player *Odd* if $v_{i+1} \in V_o$, or by $\iota_{\min}(v_{i+1}) \sqsupseteq \max_{\sqsubseteq}\{\mathsf{au}(\iota_{\min}(s), v_{i+1}) \mid (v_{i+1}, s) \in E\} \sqsupseteq \mathsf{au}(\iota_{\min}(v_i), v_{i+1})$ if $v_{i+1} \in V_e$. With $\iota_{\min}(v_n) \neq \mathsf{won}$, this provides $\iota_{\min}(v_i) \neq \mathsf{won}$ for all $i \leq n$.

Let $\mathbf{b}_0 = \_, \ldots, \_$ be the minimal element of $\mathbb{W}$, and $\mathbf{b}_{i+1} = \mathsf{au}(\mathbf{b}_i, v_{i+1})$. Then $\mathbf{b}_0 \sqsubseteq \iota_{\min}(v_0)$, and the monotonicity of au in the first element inductively provides $\mathbf{b}_i \sqsubseteq \iota_{\min}(v_i)$ for all $i \leq n$. Thus $\mathbf{b}_n \neq \mathsf{won}$, and player *Even* loses the update game. □

8 Complexity

We use natural representation for the set of colours as integers written in binary, encoding the _ as 0. The first observation is that the number of individual lift operations is, for each vertex, limited to $|\mathbb{W}|$, the number of witnesses.

Lemma 18 *For each vertex, the number of lift operations is restricted to $|\mathbb{W}|$. The overall number of lift operations is restricted to $|V| \cdot |\mathbb{W}|$. The number of lift operations an edge (or: source or target vertex of an edge, respectively) is involved in is restricted to $|\mathbb{W}|$. Summing up over all edges and over the number of lift operations, their target* or *source vertex is involved in amounts to $O(|E| \cdot |\mathbb{W}|)$.*

A simple implementation can track, for each vertex, the information which position in the witness is the next one that would need to be updated to trigger a lift along an edge, and, using a binary representation in line with $\preceq$, which bit in

the representation of this position has to change to consider triggering an update. (Intuitively the most significant bit that separates the current value from the next value to trigger an update.)

Obviously, the most expensive path to $\iota_{\min}$ is for each position to go through all values of $|\mathbb{W}|$ in this case. But in this case, tracking the information mentioned in the previous section reduces the average cost of an update to $O(1)$. The information that we store for this is, for each vertex, the current witness that represents its current value before and after executing the antagonistic update, and the next value that would lead to a lift operation on the antagonistic value.

For each incoming edge, the position and bit that need to be increased to trigger the next lift operation for this vertex are also stored.

Example 1 We look at a vertex v with one outgoing edge to its successor vertex s. We have 7 different colours, 2 through 8. Vertex v has colour 2.

We use a representation that follows the $\preceq$-order and thus maps _ to 0, 7 to 1, 5 to 2, 3 to 3, 2 to 4, 4 to 5, 6 to 6, 8 to 7. That is, we use the numbers 0 to 7 in order to represent the ranking of the colours (and _) in the $\preceq$-order.

Assume that s has currently a witness $\mathbf{b} = b_2, b_1, b_0 = 6, \_, 2$ attached to it, represented as $\widetilde{\mathbf{b}} = \widetilde{b}_2, \widetilde{b}_1, \widetilde{b}_0 = 6, 0, 4$.

To obtain a witness for v, we calculate $\mathbf{c} = \mathsf{au}(\mathbf{b}, v) = 6, 5, 2$, which is represented as $\widetilde{\mathbf{c}} = \widetilde{c}_2, \widetilde{c}_1, \widetilde{c}_0 = 6, 2, 4$. The next higher value $\mathbf{a} \sqsupset \mathbf{b}$ such that $\mathsf{au}(\mathbf{a}, v) \sqsupset \mathsf{au}(\mathbf{b}, v)$ is $\widetilde{\mathbf{a}} = \widetilde{a}_2, \widetilde{a}_1, \widetilde{a}_0 = 6, 2, 4$.

The lowest position i with $\widetilde{a}_i > \widetilde{b}_i$ is for position $i = 1$, and the difference occurs in the middle bit ($\widetilde{a}_1 = 2 = 010_2$ and $\widetilde{b}_1 = 0 = 000_2$).

For the edge from v to s, we can store after the update that we only need to consider an update from s if it increases at least the position b_1 of the witness for s. If b_1 is changed, we only have to consider the change if the update is at least to the value represented as 2 ($\widetilde{b}'_1 \geq 2$), and thus $b'_1 \succeq 5$. For all smaller updates of the witness of s, no update of the witness of v needs to be considered.

Assuming members of $\mathbb{W}$ can be concisely represented in a way where the $\preceq$-order, successor, etc., can be efficiently found (as is the case for most of our results), we have the following theorem.

Theorem 4 *For a parity game with n vertices and m edges, the algorithm can be implemented to run in $O(m \cdot |\mathbb{W}|)$ time and $O(n \cdot \log |\mathbb{W}| + m \log\log |\mathbb{W}|)$ space.*

Note that the $\log\log |\mathbb{W}|$ information per edge is only required to allow for a discounted update cost of $O(1)$. It can be traded for a $\log |\mathbb{W}|$ increase in the running time. This leaves the estimation of $|\mathbb{W}|$.

To improve the complexity especially in the relevant lower range of colours, we first look into reducing the size of $\mathbb{W}$, and then look into keeping the discounted update complexity low. We make three observations that can be used to reduce the size of $\mathbb{W}$; they can be integrated in the overall proof, starting with the raw and basic update steps.

The first observation is that, if the highest colour is the odd colour $o_{\max}$, then we do not need to represent this colour: if $\phi(v) = o_{\max}$ and $\mathbf{b} \neq \mathsf{won}$, then $\mathsf{up}(\mathbf{b}, v)$ contains only _ and $o_{\max}$ entries. Moreover, _ and $o_{\max}$ entries behave in exactly the same way. This is not surprising: $o_{\max}$ is the most powerful colour, and a state with colour $o_{\max}$ cannot occur on a winning cycle.

The second observation is that, if the lowest colour is the odd colour $o_{\min}$, then we can ignore it during all update steps without violating the correctness arguments. (In fact, this colour cannot occur at all when using the update rules suggested in Calude et al. [6].)

Finally, we observe that, for the least relevant entry b_0 of an witness $\mathbf{b}$, it does not matter if this entry contains _ or an odd value. We can therefore simply not use odd values at this position. (Using the third observation has no impact on the complexity of the problem, but still approximately halves the size of $\mathbb{W}$, and is therefore useful in practice.)

We call the number of different colours, not counting the maximal and minimal colour if they are odd, the number r of *relevant colours*.

Lemma 19 *For a parity game with r relevant colours and e vertices with even colour, and thus with length $l = \lceil \log_2(e+1) \rceil$ of the witnesses, $|\mathbb{W}| \leq 1 + \sum_{i=0}^{l} \binom{l}{i} \cdot \binom{i+r-1}{r-1}$.*

Proof The 1 refers to the dedicated value won. For the other witnesses, the values can be obtained by considering the number i of integer entries. For i integer entries, there are $\binom{l}{i}$ different positions in the witnesses that could hold these i integer values. Fixing these positions, there are $\binom{i+r-1}{r-1}$ ways to assign non-increasing values from the range of relevant colours. (e.g. these can be represented by a sequence of i white balls and $r-1$ black balls. The number of white balls prior to the first black ball is the number of positions assigned the highest relevant colour, the number of white balls between the first and second black ball is the number of positions assigned the next lower colour, etc.) □

This allows for two easy estimations of the size of $|\mathbb{W}|$: If the number c of colours is small (especially if c is constant), then we can use the coarse estimate $|\mathbb{W}| \in O\left(e \cdot \binom{l+r-1}{l}\right)$.

In particular, we get the following complexity for a constant number of colours.

Theorem 5 *A parity game with r relevant colours, n vertices, m edges, and e vertices with even colour can be solved in time*

$O\big(e \cdot m \cdot (\log(e)+r)^{r-1}/(r-1)!\big)$ *and space* $O\big(n \cdot \log(e) \cdot \log(r) + m \cdot \log(\log(e) \cdot \log(r))\big)$.

We use that the length $l = \lceil \log_2(e+1) \rceil$ of the witnesses is logarithmic in e.

This also provides us with a strong fixed parameter tractability result: when we fix the number of colours to some constant c, we maintain a quasi-bi-linear complexity in the number of edges and the number of vertices. If we fix, e.g. a monotonously growing quasi-constant function qc (like the inverse Ackermann function), then Theorem 5 shows that, as soon as $\mathsf{qc}(n) \geq c$, and thus almost everywhere and in particular in the limit, have $(l+r)^{r-1}/(r-1)! \leq (\log_2 n)^{\mathsf{qc}(n)}$, or $(l+r)^{r-1}/(r-1)! \leq \mathsf{qc}(n)^{\log_2(\log_2(n))}$ if $\log_2(\mathsf{qc}(n) \geq c)$.

Corollary 5 *Parity games are fixed parameter tractable, using the number of colours as their parameter, with complexity* $O\big(m \cdot n \cdot \mathsf{qc}(n)^{\log\log n}\big)$ *for an arbitrary quasi-constant* qc*, where m is the number of edges and n is the number of vertices.*

For a "high" number of colours, we can improve the estimation: if $r \geq l^2$, then the case $i = l$ dominates the overall cost, such that $|\mathbb{W}| \in O\Big(\binom{l+r-1}{l}\Big)$.

Theorem 6 *For a parity game with r relevant colours, m edges, and e vertices with even colour, and thus length* $l = \lceil \log_2(e+1) \rceil$ *of the witnesses, and* $h = \lceil 1 + \frac{r-1}{l} \rceil$*, one can solve the parity game in time* $O(m \cdot h \cdot e^{1+c_{1.45}+\log_2(h)})$*, and in time* $O(m \cdot h \cdot e^{c_{1.45}+\log_2(h)})$ *if* $r > l^2$.

We use the constant $c_{1.45} = \lim_{h\to\infty} \log_2(1+1/h) \cdot h = \log_2 \mathsf{e} < 1.45$, where $\mathsf{e} \approx 2.718$ is the Euler number; using that $(1+1/h)^h < \mathsf{e}$ and thus $\log_2(1+1/h) \cdot h < c_{1.45}$ holds for all $h \in \mathbb{N}$.

Proof To estimate $\mathbb{W}$, we again start with analysing the size of $\binom{l+r-1}{l}$.

We note that $l + r - 1 \leq h \cdot l$, such that we can estimate this value by drawing l out of $h \cdot l$.

The number of all ways to choose $l = \lceil \log(e+1) \rceil$ out of $h \cdot l$ numbers can, by the Wikipedia page on binomial coefficients and the inequality using the entropy in there (also can be found in [2]), be bounded by

$$
\begin{aligned}
&2^{(\log_2(e)+1)\cdot h\cdot((1/h)\cdot\log_2(h)+((h-1)/h)\cdot\log_2(h/(h-1)))} \\
&\quad = 2^{(\log_2(e)+1)\cdot(\log_2(h)+\log_2(1+1/(h-1))\cdot(h-1))} \\
&\quad = (2e)^{\log_2(h)+(\log_2(1+1/(h-1)))\cdot(h-1))} \\
&\quad \leq (2e)^{c_{1.45}+\log_2(h)} \in O\big(h \cdot e^{c_{1.45}+\log_2(h)}\big).
\end{aligned}
$$

The estimation uses that $\log(1+1/(h-1))\cdot(h-1) < c_{1.45}$ holds for all $h \in \mathbb{N}$.

Theorem 4 now provides $O(m \cdot h \cdot e^{1+c_{1.45}+\log_2(h)})$ time bound. If the number of colours is high ($r > l^2$), then we observe that $|\mathbb{W}| \leq 1 + \sum_{i=0}^{l} \binom{l}{i} \cdot \binom{i+r-1}{i} \in O\Big(\binom{l+r-1}{l}\Big)$ holds, as the sum is dominated by $\binom{l}{l} \cdot \binom{l+r-1}{l}$. This allows for the second estimate $O(m \cdot h \cdot e^{c_{1.45}+\log_2(h)})$ of the running time when $r > l^2$ holds. □

This allows for identifying a class of parity games that can be solved in polynomial time.

Corollary 6 *Parity games where the number c of colours is logarithmically bounded by the number e of vertices with even colour* ($c \in O(\log e)$) *can be solved in polynomial time.*

9 Witness length as an index of complexity

Lehtinen [26] introduces the notion of *register index* of a parity game, which is a natural number associated to a parity game and is intended as a measure of the complexity of solving it. Lehtinen compares the register index to two other previously proposed measures of complexity for parity games: *entanglement* and the *Rabin index*. Moreover, the author proves that the register index of every parity game is at most $\log n + 1$, and that a parity game of register index k with n vertices is equivalent to a parity game that has $4n|C|^k$ vertices with $2k+1$ colours, where C is the set of colours of the original parity game. Together with, e.g. Theorem 4, this yields an alternative proof of the quasi-polynomial time solvability of parity games. The space complexity of Lehtinen's method is rather high, as it works by reducing a parity game to a much bigger one, where the magnitude of the increase depends on the register index of the parity game under consideration.

In this section, we propose two new complexity measures that are naturally derived from the update statistics games discussed in this paper. We call these the *forward* au*-index* and the *backward* au*-index*. These indices are informally defined as the minimum length ℓ of the witness that we need to track in order to establish whether one of the two players wins the antagonistic update game (using, respectively, forward and backward evaluation of the witness). For the backward au-index, we show that we may then slightly adapt our progress measure algorithm such that (roughly stated) it lifts only up to witnesses that do not use the indices exceeding ℓ. Using Theorem 4, taking for $|\mathbb{W}|$ the upper bound $|C|^\ell$, we establish that we can solve such a game in time $O(n\ell m|C|^\ell)$ and space $O(n\ell^2 \log|C| + m\ell \log \ell + m\ell \log\log|C|)$. For the forward au-index, we relate it to the register index by showing that it is always at least as high as the register index. As the forward au-index is at most $\lfloor \log n \rfloor + 1$, this yields an alternative proof of the upper bound of $\lfloor \log n \rfloor + 1$, which was shown in [26] using an entirely different argument.

Let us first precisely define the antagonistic update index. This index is naturally obtained by considering that the roles of players even and odd can be swapped, and that the antagonistic update game defined in Sect. 6 can be run relative to player *Odd* just as well as player *Even*. We refer to these two games as the *even antagonistic update game* and the *odd antagonistic update game*. If player *Even* cannot force a win in the even antagonistic update game, then player *Odd* has a strategy such that every play results in a witness $\mathbf{b}$ where $\mathsf{value}(\mathbf{b}) \leq e$. In particular, something even stronger may be the case: The odd player may have a strategy such that every play results in a witness with a value *much lower* than e. When the odd player uses such a strategy, every play will result in a witness where the highest indices of the witness are always set to _. A symmetric situation occurs in the odd antagonistic update game when the odd player cannot force a win. The *antagonistic update index* is defined as the least index ℓ such that the winning player has a strategy such that in either the odd or even antagonistic update game it holds that every play results in a witness $\mathbf{b}$ where $b_\ell = \_$.

Definition 3 Let $\mathcal{P}$ be a parity game with a specified starting vertex. The *backward (forward) antagonistic update index* (au-*index)* ℓ of $\mathcal{P}$ is defined as follows.

- If player *Odd* wins $\mathcal{P}$, then ℓ is defined as the least index for which odd has a strategy σ such that for each prefix p of each play ρ that is consistent with σ, the backward (forward) evaluation of p using the antagonistic update rule au results in a witness $\mathbf{b}$ where $b_\ell = \_$.
- Similarly, if player *Even* wins $\mathcal{P}$, then ℓ is defined as the least index for which even has a strategy σ such that for each prefix p of each play ρ that is consistent with σ, the backward (forward) evaluation of ρ using the odd player equivalent of the antagonistic update rule **au** results in a witness $\mathbf{b}$ where $b_\ell = \_$.

The *global backward (forward)* au-*index* of a parity game is defined as the highest backward (forward) au-index with respect to any starting vertex.

Let us first give an example of a parity game for which it holds that there are many vertices and colours, but has a low global forward and backward au-index: Let our parity game consist of an arbitrary number of disjoint directed 4-cycles, all with disjoint colours such that the ith cycle has colours $4i, 4i+1, 4i+2, 4i+3$ on its respective vertices, so that the maximum colour of each cycle is odd. Since each vertex has a single outgoing arc, in this parity game it is irrelevant which player controls which vertex. It can now straightforwardly be shown that when playing the forward antagonistic update game from any starting vertex, the values of the witness at indices exceeding 1 always remain _. Due to the various details in the definition of the antagonistic update rule, showing this formally would require a somewhat analysis, although the core reason for this being true lies in the fact that the dominating colour of the play is encountered once every four vertices in any prefix of any play of this parity game. A similar argument can moreover be made for the backward version of the antagonistic update game. This leads us to the conclusion that the global forward and backward au-index of this family of example parity games is 2.

We will use the symbol ℓ to refer to the forward or backward au-index of a parity game. By definition of ℓ, there exists a strategy for the winning player that prevents the losing player's update statistics from reaching a witness in which any index of ℓ or higher is used. We refer to such a strategy as a *certificate strategy* for the (forward or backward) au-index.

Definition 4 Let $\mathcal{P}$ be a parity game with a specified starting vertex and let ℓ (ℓ') be the forward (backward) au-index of $\mathcal{P}$. A *certificate strategy* for the forward (backward) au-index is a strategy τ of the winning player such that for all plays ρ in agreement with τ, for every prefix p of ρ, the forward (backward) evaluation of p according to the losing player's antagonistic update rule results in a witness $\mathbf{b}$ satisfying $b_i = \_$ for all $i \geq \ell$ ($i \geq \ell'$).

The following theorem shows that parity games with a low global backward au-index can be solved in a time that is much faster than the quasi-polynomial worst-case bounds given in Sect. 8.

Theorem 7 *A parity game $\mathcal{P}$ where the colour set is C and the global backward* au-*index is ℓ can be solved in time $O(n\ell m|C|^\ell)$ and space $O(n\ell^2 \log|C| + m\ell \log \ell + m\ell \log\log|C|)$.*

Proof If we know ℓ, we may run the value iteration algorithm of Sect. 7, with the following adaptation: when in an iteration i we encounter a vertex v with $\iota_i(v)_\ell \neq \_$, we lift the vertex immediately to the witness won in iteration $i+1$. Observe that the resulting progress measure at iteration $i+1$ is still less than $\iota_{\min}$ with respect to the partial order in the lattice defined by pointwise comparison, thus the final progress measure found by the algorithm will still be the minimal consistent progress measure. Observe that since all witnesses encountered in the algorithm have the value _ on indices at least ℓ, the number of witnesses $|\mathbb{W}|$ with which the algorithm effectively works is $(|C|+1)^\ell$, so it runs in time $O(nm|C|^\ell)$ and space $O(n\ell \log|C| + m\log\ell + m\log\log|C|)$ by Theorem 4.

If ℓ is unknown, we may proceed as follows. We run the above adaptation of the progress measure algorithm iteratively for all possible values of ℓ, starting at $\ell = 1$ and incrementing ℓ at each iteration. In each iteration, we run two instances I_{even} and I_{odd} of this algorithm in parallel: The witnesses of I_{even} are witnesses of the even antagonistic

update game and the witnesses of I_{odd} are witnesses of the odd antagonistic update game. Let the respective outputs of these two algorithms be $\iota_{\min,\mathsf{even}}$ and $\iota_{\min,\mathsf{odd}}$, which are, by definition, the minimum consistent progress measures corresponding to the even and odd antagonistic update games under the assumption that the global au-index of $\mathcal{P}$ is ℓ.

It is now easy to check whether we have run our progress measure algorithm for the correct choice of ℓ: from the output of the algorithms we may define two sets of vertices V_{odd} and V_{even}, where V_{odd} is the set of vertices v such that $\iota_{\min,\mathsf{even}} \neq \mathsf{won}$, and V_{even} is the set of vertices v such that $\iota_{\min,\mathsf{odd}} \neq \mathsf{won}$. If ℓ is chosen correctly, then $\iota_{\min,\mathsf{even}}$ is the minimum consistent progress measure with respect to the even antagonistic update statistics, so V_{odd} is the set of vertices where player *Odd* wins and the proof of Lemma 17 provides a constructive linear time algorithm for obtaining a memoryless winning strategy σ for player *Odd* on V_{odd}. Likewise, from $i_{min,odd}$ we may obtain a memoryless winning strategy τ for player even on V_{even}. If $(V_{\mathsf{even}}, V_{\mathsf{odd}})$ is not a partition of V or if (σ, τ) is not a valid pair of memoryless winning strategies, then ℓ is not the correct value of the global au-index, so we may increment ℓ and repeat the process. Otherwise, we have found the correct pair of memoryless winning strategies and thereby solved the parity game and found the correct value of ℓ.

For completeness, we mention that it is easy to algorithmically check whether a memoryless strategy for a given player on a given set of vertices V' is winning: just remove from the graph all the arcs that are controlled by the player but that are not in the strategy. Clearly, the strategy is winning if and only if the subgraph induced by V' contains no cycle where the highest priority corresponds to the opposing player.

The time complexity of this procedure is as follows. Let ℓ now be the correct progress measure. This algorithm runs through 2ℓ executions of the adapted version of our progress measure algorithm, where in each iteration it uses at most $(|C|+1)^{\ell}$ witnesses. Therefore, by Theorem 4, the complete procedure takes time $O(n\ell m|C|^{\ell})$ and space $O(n\ell^2 \log|C| + m\ell \log \ell + m\ell \log\log|C|)$. □

Note that the above proof also implies that there always exist *memoryless* strategies for the players that are certificate strategies for the global backward au-index of a parity game, which can be found within the specified time and space bounds.

Lehtinen's *register index* provides an alternative measure of complexity for solving a parity game, and it is defined as follows. First, we define a *register game* with k registers on a given parity game. In such a game, play proceeds as usual, and additionally one of the players, say even, has some control over k registers, each containing a colour or the value -1 and are re-ordered at each step of the game. Initially all registers contain the value -1 and are sorted arbitrarily. Each time the token is moved (either by the even or the odd player) to a new vertex having some colour c, the content of each register r is set to $\max\{c, b\}$, where b is the current content of register r. Each time the token moves, the even player may additionally choose one of these registers and reset it to -1. At each step, registers are re-ordered non-decreasingly according to their content, breaking ties arbitrarily, effectively meaning that the register that is reset gets taken out of the ordering and gets inserted at the lowest position (i.e. position 1) in the ordering. The winning condition is as follows, assuming player *Even* controls the registers (otherwise, switch the roles of even and odd): Player Odd wins if player *Even* only resets registers finitely often. If player *Even* resets registers infinitely often, then let r be the highest rank in the ordering at which registers are reset infinitely often and let x be the number of times that a register with an odd number is reset at rank r. Player Even wins the register game if and only if x is finite.

The register index for a parity game (with specified starting position) is defined as the least number of registers k such that the winning player of the parity game wins the k-register game in which the winning player is in control of the registers.

Lehtinen shows that the k-register game, for any k, is in turn equivalent to a parity game of $2k+1$ colours and $4n|C|^k$ vertices. The author also shows that $k \leq \log(n+1)$ and these facts together imply that parity games can be solved in quasi-polynomial time.

The register index has been introduced with the purpose of providing a measure of complexity of solving a parity game. Therefore, we are interested in the question whether the register index relates in some way to the required size of the witness generated by our antagonistic update rule, and to that end we prove next that the register index is always at most the value of both the forward and backward au-indices.

The first lemma we need involves the witnesses generated by forward evaluation of the play. It shows the following, informally stated: In the forward antagonistic update statistics game corresponding to the losing player, if we inspect the value at the largest index of the witness where update rule 2 (according to Definition 1) is applied infinitely often with a colour of the winning player, the highest colour that is encountered infinitely often at that index is the winning colour of the play.

Lemma 20 *Let ρ be a play on a parity game and consider the infinite sequence of witnesses resulting from the sequence of updates of the losing player's antagonistic update statistics. Let c be the highest colour that occurs infinitely often in ρ, let j be the highest index of the witness that is modified infinitely often, and let c' be the highest colour such that update rule 2 (recall Definition 1) is applied infinitely often on index j of the witness, either on witness* **b** *directly or on an antagonistic*

choice $\mathbf{d} \sqsupset \mathbf{b}$ *(i.e. resulting in value* c' *on the* j*th index after applying the update rule). Then* $c' = c$.

Proof We may assume without loss of generality that we are given a parity game in which the winning player is odd, and we consider evaluating a play $\rho = (v_1, v_2, \ldots)$ with the even antagonistic update statistics. The play ρ induces an infinite sequence of witnesses $(\mathbf{b}_1, \mathbf{b}_2, \ldots)$, where $\mathbf{b}_i$ is the witness obtained after forward evaluation of the prefix $(v_1, \ldots, v_i)$ of ρ, starting from witness $(\_, \ldots, \_)$.

The definition of the antagonistic update rule and Definition 1 together with Corollary 1 imply that at each step i of the play, the witness $\mathbf{b}_i$ is updated by selecting a $\sqsubseteq$-higher witness $\mathbf{d}_i$ and subsequently applying update rule 1 or 2 on it. From Corollary 2, it follows that if update rule 1 is applied on $\mathbf{d}_i$, then for the resulting witness $\mathbf{b}_{i+1}$ it holds that $\mathbf{b}_{i+1} \sqsupset \mathbf{b}_i$.

Our sequence of witnesses resulting from ρ never attains the value won, which implies that for generating the infinite sequence of witnesses $(\mathbf{b}_1, \mathbf{b}_2, \ldots)$, update rule 2 is used infinitely often. (If update rule 2 would only be used finitely often, then there would be some point in the sequence after which solely update rule 1 is used, but by the strong monotonicity we just established for the latter operation, that would mean that the value won is reached after a finite number of iterations, which is a contradiction.)

Let c be the highest colour occurring infinitely often in ρ, which must be an odd colour. Let j be the maximal index for which update rule 2 is applied at index j infinitely often. Note that the definition of j implies that there is a point i in the play such that in the suffix of the play from element i onward, no index exceeding j is ever modified: First there certainly exists a point i' such that from step i' onward rule 2 is never applied at any index exceeding j. Thus from i' onward, any modification of the witness at any index exceeding j must be at steps i'' such that rule 1 is used on $d_{i''}$. We then combine the following:

- the definition of the $\sqsubseteq$-relation,
- the fact that at such a step i'' there is an index exceeding j at which the value $\preceq$-increases (using the definition of update rule 1 and the definition of the antagonistic choice $\mathbf{d}_{i''}$),
- the fact that applications of rule 2 do not modify any index exceeding j,

from which we conclude that steps where update rule 1 is used, and where an index exceeding j is modified, cannot occur infinitely often. This implies the existence of i. We may also assume that there is a step after which no colour higher than c ever occurs, hence we may assume that i also satisfies that property. Next, let c' be the highest colour such that rule 2 is applied infinitely often on c' at index j. We may assume that there is a step after which rule 2 is never applied at index j on any colour higher than c', and we assume that i also satisfies that property. We refer to steps where rule 2 is applied on c' at index j as *dominant*. Finally, we assume that at iteration i, the witness $\mathbf{b}_i$ does not hold value _ at index j and a dominant step has occurred at least once. Note that this implies that at every point after i, index j does not hold the value _ and holds either an even value or an odd value of c' or less. We restrict our attention to the suffix of ρ starting at step i.

Suppose for contradiction that $c' < c$. Then, after each step $i' \geq i$ where c occurs, there is a colour $c'' \geq c$ at index j of the witness, which must be an even colour, as we established that index j cannot hold odd colours greater than c' from step i onward. Since after such a step eventually a dominant step $i'' > i'$ occurs, this means that in between i' and i'', the value at index j decreases to a value $c'' < c'$, which can only be achieved through some minimal step i''' with $i' \leq i''' < i''$, where update rule 1 is applied at index j and $\mathbf{b}_{i'''} = \mathbf{d}_{i'''}$. Since index j of the witness contains an even value of c'' at step i', we conclude that the value of index j is even and at least c'' at step i'''; a contradiction with the fact that update rule 1 is only applied on indices where the witness contains a non-even value. Therefore, $c = c'$, which proves the claim.[1] □

The following lemma shows that the register index of a parity game is at most the forward au-index ℓ. Its proof makes use of Lemma 20 in order to design a strategy for the associated register game with ℓ registers controlled by the winning player.

Theorem 8 *Let* $\mathcal{P}$ *be a parity game with specified starting vertex and let* ℓ *be the forward* au*-index of* $\mathcal{P}$*. Then the register index of* $\mathcal{P}$ *is at most* ℓ.

Proof We show that the winning player of $\mathcal{P}$ can force a win in the register game on $\mathcal{P}$ with ℓ registers. Assume that player *Odd* is the winning player, and let τ be a certificate strategy of player *Odd* for the forward au-index. We define a strategy for player *Odd* in the register game over $\mathcal{P}$, where the odd player controls ℓ registers: Player Odd moves the token through the graph exactly as τ prescribes. For each move, player *Odd* tracks how the witness resulting from the even forward antagonistic update rule is updated. Depending on the antagonistic update rule applied at the last move, player *Odd* resets a register as follows, where we recall that indices of the witness are numbered starting from 0 and registers are numbered starting from 1:

- If update rule 2 was applied, and the colour c to which the token was moved was odd, then let $j' < \ell$ be the index of

[1] We recall Definition 1, which state when rules 1 and 2 are defined to be applied.

the witness that was updated to colour c. The odd player resets register $j' + 1$.
- Otherwise, (if the colour c to which the token was moved was even, or if update rule 1 was applied), the odd player does not reset any register.

Using Lemma 20, it is quite straightforward to establish that this strategy forces the odd player a win in the register game: Consider any play ρ that agrees with τ and let the index $j < \ell$ be defined as in Lemma 20. The lemma states that index j of the witness is updated infinitely often to the highest infinitely often occurring colour of ρ by means of applications of update rule 2. By our definition of the strategy of player *Odd* in the register game, register $j + 1$ is reset infinitely often with the highest infinitely often occurring colour in it, which is an odd colour. Moreover, no register exceeding $j + 1$ is reset infinitely often by definition of j.

It remains to prove that register $j+1$ is not reset an infinite number of times when it contains an even colour. To prove that, we define t as the earliest point in the play ρ such that up to step t there has been at least one update of type 2 at index j of the witness on the highest infinitely often occurring colour, and moreover, from t onward there occurs no colour exceeding the highest infinitely often occurring colour, and no index of the witness that exceeds j is ever modified. We now need the following insight about the play ρ.

Proposition 1 *Suppose that $x \geq t$ is a time step at which an even colour c is held at index j of the witness, and let $x' > x$ be the earliest time step at which update rule 2 is used at index j on an odd colour c'. Then $c' > c$.*

Proof By definition of the antagonistic update rule and by the fact that in between x and x', update rule 2 is not used at index j on an odd number, we conclude that any update of the witness in between x and x' either leaves the value at index j unchanged or $\preceq$-increases the value of the witness at index j to a higher even number. Therefore, at time step x' update rule 2 is applied on an odd colour $c' > c$.

Let i and i' be two steps where $t \leq i < i'$ such that τ prescribes to reset register $j+1$ in steps i and i', and register $j + 1$ is not reset in between i and i'. Let c' be the colour occurring at step i'. Suppose for contradiction that there is a even colour c'' occurring in between steps i and i' such that $c'' > c'$, and let i'' be the corresponding step at which this happens. Index j of the witness at step i'' (after updating it with colour c'') must then contain a colour of c'' or higher, but by the proposition above that would imply that $c' > c''$; a contradiction. We therefore establish the following: For any colour c'' occurring after t in between two consecutive type 2 updates at index j of the witness on odd colours c and c', respectively, it holds that c'' is not even if $c'' > c'$. From the definition of the register resetting strategy τ it then directly follows that after time step t, the number of times register $j+1$ is reset with an even number containing it is 0, and thus the total number of times that register $j + 1$ is reset with an even number containing it is t, which is finite.

Therefore, register $j+1$ is the highest register that is reset infinitely often and register $j+1$ is reset only a finite number of times with an even number contained in it. This implies that player *Odd* is guaranteed to win the register game with strategy τ. The number of registers used is at most the number ℓ of indices of the witness that are modified throughout the play, which proves the claim. □

The above bound additionally provides us with an alternative proof of the fact that the register index is at most $\lfloor \log n \rfloor + 1$: Because the highest index of a witness does not exceed $\lfloor \log n \rfloor$, the forward au-index ℓ is at most $\lfloor \log n \rfloor + 1$, which means that the register index is at most $\lfloor \log n \rfloor + 1$ by Theorem 8.

Corollary 7 *The register index of a parity game with n vertices is at most $\lfloor \log n \rfloor + 1$.*

In relation to the register index, we leave open a few interesting problems: First of all, we wonder if it is possible to complement our lower bound on the forward au-index by a corresponding upper bound in terms of the register index. Secondly, we hope to achieve a similar result for the backward au-index, which would be insightful with respect to the value iteration algorithm of Sect. 7, where the progress measure was defined using backward evaluation. The challenge for obtaining such a result seems to be to obtain useful insights by which we can conveniently relate a *backward* play (forming the witness) to the corresponding *forward* play (forming the register configuration).

10 Lower bounds

In this section, we introduce a family of examples, on which the basic update game from [6] is slow. (Recall that these original rules restrict the use of Lemma 3 to even colours. Adjusting the example is not hard, but effectively disallows to make effective use of b_0.)

The example is a single player game, which is drawn best as a ring. In this example, the losing player, player *Odd*, can draw out his loss. The vertices of the game have name and colour $1, \ldots, 2n$. They are all owned by player *Odd*. There is always an edge to the next vertex (in the modulo ring). Additionally, there is an edge back to 1 from all vertices with even name (and colour). Figure 2 depicts the example for $n = 2$.

Obviously, all runs are winning for player *Even*. We show how player *Odd* can, when starting in vertex 1, produce a play, such that forward updates produce all witnesses that use only _ and even numbers.

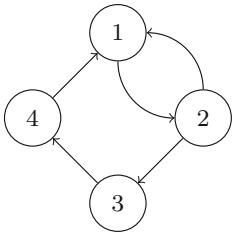

Fig. 2 The lower bound example for $n = 2$

We first observe that every value $2i-1$ is overwritten after the next move in a play by $2i$ in a witness **b**.

The strategy of player *Odd* to create a long path is simple. We consider three cases.

If, in the current witness $\mathbf{b} = b_k, \ldots, b_0$, we have $b_0 = \_$ and the token is at a position $2i$, then moving to 1, and thus next to 2, results in the next witness without odd entries larger than **b**.

If $b_0 \neq \_$, then we have that $b_0 = 2i$, and **b** has no smaller entries than $2i$. If all of these entries are consecutively on the right of **b**, then we obtain the next witness without odd entries larger than **b** by going through $2i+1$ to $2i+2$. Player Odd therefore chooses to continue by moving the token to vertex $2i+1$ in this case.

Otherwise, there is a rightmost $b_j = \_$, such that right of it are only entries $2i$ (for all $h < j$, $b_h = 2i$), and there is also a $2i$ value to the left (for some $h > j$, $b_h = 2i$). Then the next witness without odd entries larger than **b** is obtained by replacing b_j by 2 and all entries to its right by _. This can be obtained by going to vertex 1 and, subsequently, to vertex 2. Player Odd therefore chooses to continue by moving the token to vertex 1 in this case.

11 Implementation

We implemented our algorithm in C++ and tested its performance on Mac OS X with 1.7 GHz Intel Core i5 CPU and 4 GB of RAM. We then compared it with the small progress measure algorithm [22], Zielonka's recursive algorithm [41], the classic strategy improvement algorithm [39] as implemented in the PGSolver version 4.0 [14,16], and the implementation [37] of an alternative recently developed succinct progress measure algorithm from [20]. We tested their performance, with timeout set to two minutes, on around 250 different parity games of various sizes generated using PGSolver. These examples include the following classes.

- Friedmann's trap examples [17], which show exponential lower bound for the classic strategy improvement algorithm;
- random parity games of size s, ranging from 100 to 10,000 that were generated using PGSolver's command `steadygame` s `1 6 1 6` (for each s we generated ten instances);
- recursive ladder construction [18] generated using PGSolver's command `recursiveladder`.

PGSolver implements several optimisation steps before the algorithm of choice is invoked. These include SCC (Strongly Connected Component) decomposition, detection of special cases, priority compression, and priority propagation as described in [16]. To illustrate this, the small progress measures algorithm in PGSolver was able to solve all Friedmann's trap examples in 0.01 s when using these optimisations. However, without these optimisations, it failed to terminate within the set timeout of two minutes. As our aim was to compare different algorithms and not the heuristics or preprocessing steps involved, we invoked PGSolver with options "`-dgo -dsd -dlo -dsg`" to switch off some of these optimisation steps. We believe this gives a better and fairer picture of the relative performance of these algorithms. Some of these optimisations are embedded in the algorithms themselves and cannot be switched off. For example, the small progress measure algorithm implemented in PGSolver starts off with the computation of maximal values that may ever need to be considered [16]. In future, we plan to include these optimisation preprocessing techniques into our tool as well.

The more interesting results of our tests are presented in Table 1. As expected, our algorithm is outperformed by strategy improvement and recursive algorithm on randomly generated examples. Our algorithm is very fast on Friedmann's trap examples, because player *Odd* wins from all nodes and a fixed point is reached very quickly using a small number of entries in the witnesses; see Fig. 3 at the end of the paper. Finally, we tested the algorithms on the recursive ladder construction, which is a class of examples for which the recursive algorithm runs in exponential time. As expected, the small progress measure and the recursive algorithm fail to terminate for examples as small as 250 nodes. Our algorithm as well as the classic strategy improvement solved these instances very quickly. Interestingly, the worst performing algorithm is [20], which currently has the best theoretical upper bound on its running time. The most likely reason for this is that their single step of the value iteration is a lot more complicated than ours. As a result, even if fewer steps are required to reach a fixed point, the algorithm performs badly as each step is a lot slower. In conclusion, our algorithm complements quite well the existing well-established algorithms for parity games and can be faster than any of them depending on the class of examples being considered.

The implementation of our algorithm along with all the examples that we used in this comparison is available at https://cgi.csc.liv.ac.uk/~dominik/parity/.

Further investigations were done to compare the above implementation of our *basic* au*-value iteration algorithm* of Sect. 7 with the *modified* au*-value iteration* algorithm of

Table 1 Running times (in seconds) of the four algorithms tested: the quasi-polynomial time (QPT) algorithm presented in this paper, small progress measure (SPM), Zielonka's recursive algorithm (REC), the classic strategy improvement (CSI), and the implementation [37] of the quasi-polynomial time algorithm (JL'17) from [20]

Example class	Nodes	Colours	QPT	SPM	REC	CSI	JL'17 [20,37]
`steadygame`	100	100	min: 0.01	min: 0.01	min: 0.01	min: 0.01	min: 10.16
			max: 0.02	max: 0.02	max: 0.01	max: 0.01	max: –
`steadygame`	200	200	min: 0.01	min: 0.01	min: 0.01	min: 0.01	min: –
			max: 0.09	max: 0.06	max: 0.01	max: 0.03	max: –
`steadygame`	1000	1000	min: 0.09	min: 1.55	min: 0.01	min: 0.14	min: –
			max: 1.51	max: 1.67	max: 0.04	max: 0.23	max: –
`steadygame`	5000	5000	min: 1.51	min: 41.49	min: 0.23	min: 1.56	min: –
			max: 102	max: –	max: 0.44	max: 4.12	max: –
`steadygame`	10, 000	10, 000	min: 5.1	min: –	min: 0.68	min: 3.07	min: –
			max: –	max: –	max: 1.89	max: 8.25	max: –
`Friedmann's trap`	77	66	0.01	–	0.01	0.26	–
`Friedmann's trap`	230	120	0.01	–	0.01	22.72	–
`Friedmann's trap`	377	156	0.01	–	0.01	–	–
`recursive ladder`	250	152	0.01	–	–	0.01	0.66
`recursive ladder`	1000	752	0.02	–	–	0.01	–
`recursive ladder`	25, 000	15, 002	0.45	–	–	0.56	–

Entry "–" means that the algorithm did not terminate within the set timeout of two minutes. For the `steadygame` examples, we state the minimum and the maximum measured execution time for the ten examples generated for each size

Sect. 9 (as described in Theorem 7). This modified algorithm takes a value ℓ, starting at $\ell = 0$ and in each iteration runs the basic au-value iteration algorithm from the perspective of both players, with witnesses of length ℓ (i.e. the progress measure used has ℓ indices at each vertex). If the two minimal progress measures that are output yield a consistent pair of winning strategies for the two players, then the algorithm terminates and this pair of strategies is output. Otherwise, ℓ is incremented and the next iteration starts. We focus on how the basic algorithm behaves on examples similarly to those from Sect. 10 and whether the improvements in the modified algorithm overcome this problem.

The experimental data below have been obtained by running our modified algorithm on a virtual machine (Solaris zone) running on an Oracle Sparc T7-1 with an 1 M7 CPU and 256 GB RAM. In the below tables, the runtime is rounded upwards to the next full number of seconds. The corresponding C programs were compiled with the option `gcc -O3 progname.c` on UNIX.

In Sect. 10, we established that the original quasi-polynomial time algorithm as well as our basic version of the au-value iteration algorithm do not perform well on certain types of games. Indeed, even a game with a large number of colours partitioned over the set of vertices into singletons is already difficult for these algorithms to handle, as Table 2 shows.

If a game has 30 even-coloured nodes, then this implementation of the basic au-value iteration algorithm tracks in the winning statistics a witness that has one index more than with 29 even-coloured nodes, which explains the runtime difference for the game with 60 nodes and 55 colours, compared to the game with 60 nodes and 40 colours. For a comparison, the modified au-value iteration algorithm of Sect. 9 has a much better performance: For this modified algorithm, the task is almost trivial (as for most other parity game solvers), even when the number of nodes is much larger than the instances used in Table 2, as shown in Table 3:

Next, let us study our algorithms on the games from Sect. 10, and let k be the number of colours for each player, so that the number of nodes and number of colours is $n = 2k$. The number of colours is also $2k$. This game is more difficult than the above and for $k = 16, 17, 18, \ldots, 30$, that is, for $n = 32, 34, 36, \ldots, 60$, the runtimes for our basic au-value iteration algorithm are given in Table 4.

Another experimental question we are interested in is whether a high register index is harmful, as Lehtinen's quasi-polynomial time algorithm [26] is an XP algorithm with respect to the register index of the game. The results of Sect. 9 (where we show that the forward au-index is always at least the register index) seem to suggest that indeed games with a high register index will be hard to solve using our progress measure algorithms. To test this hypothesis, we converted Lehtinen's examples for small games with high register index into a form where the weights are on the nodes and not the edges (which was the case in the original examples of [26]). In this family of examples, all nodes are controlled by player

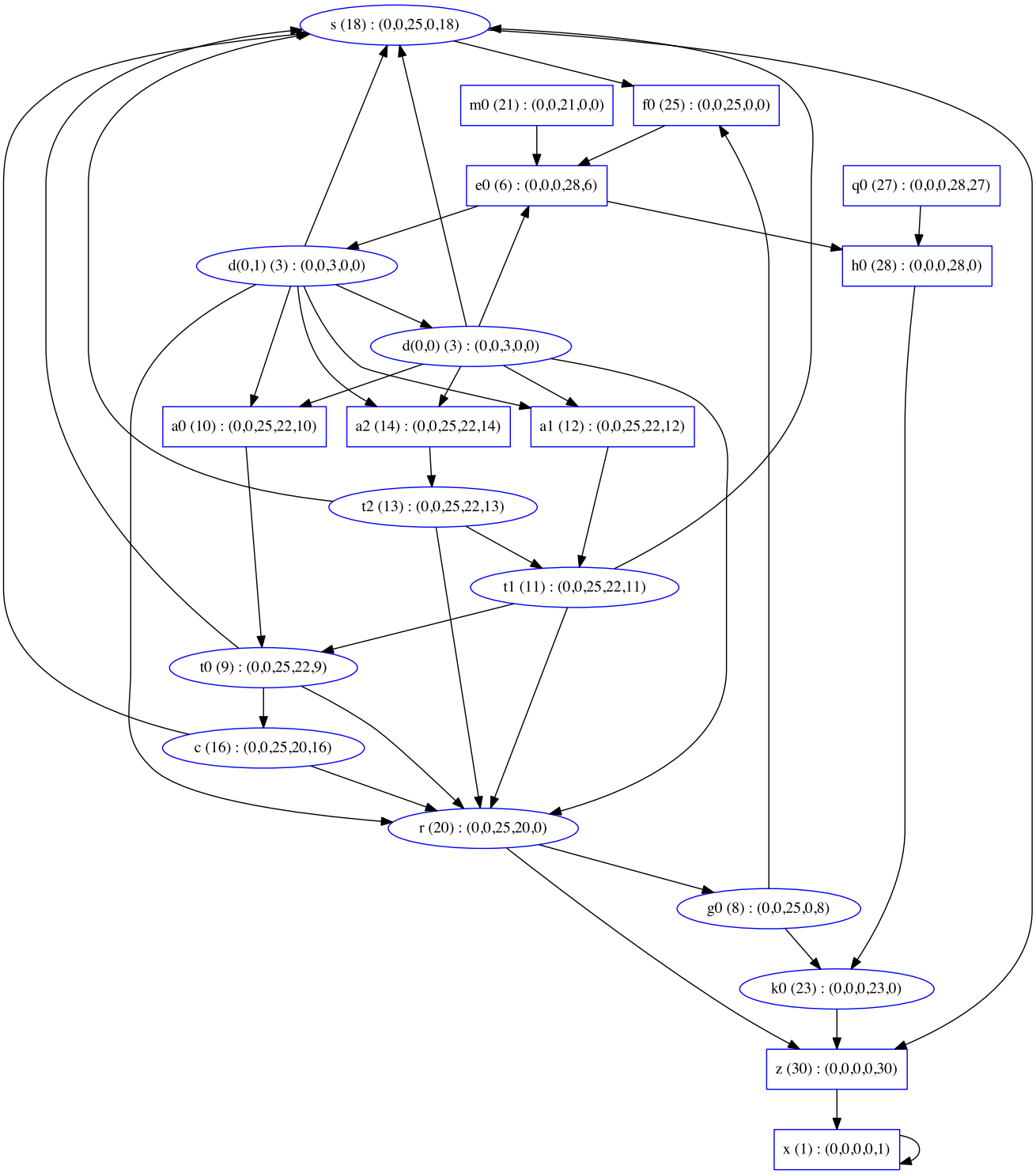

Fig. 3 The fixed point reached when using the QPT algorithm to solve the Friedmann's trap example with 20 nodes. Square nodes belong to player *Odd* and circle nodes to player *Even*. The label of a node consists of its name, followed by its colour (in parentheses), and after a colon its witness for $\iota_{\min}$ (colour figure online)

Table 2 Experimental results for our basic au-value iteration algorithm on instances consisting of singleton vertices with a large number of colours

Nodes	Colours	Even-coloured nodes	Time (s)
30	30	15	1
40	40	20	1
50	50	25	2
60	30	30	3
60	35	29	1
60	40	30	11
60	45	29	2
60	50	30	27
60	55	29	3
60	60	30	56

Table 3 Experimental results for our modified au-value iteration algorithm on instances consisting of singleton vertices with a large number of colours

Nodes	Colours	Even-coloured nodes	Time (s)
1000	1000	500	0.004
10,000	10,000	5000	0.024

Odd, and they formed by taking as a basis instance a node with a self-loop of the lowest even colour 0. Then, the following operation is performed a number of times: A copy of the current instance along with two new colours are introduced that are higher than all previous colours (one even colour and one odd colour, where the even colour exceeds the odd colour), and the two copies are connected by two edges of which the colours are the two respective newly introduced colours. Let κ be the number of times that this operation is repeated, then there are $O(2^\kappa)$ nodes and $2\kappa+1$ colours in the resulting instance. Even wins this parity game from all starting positions, as all cycles are dominated by an even colour. We refer to [26] for the precise details of this construction, where it is shown that for this instance the register index is $\kappa+1$, where indeed, κ is the number of repetitions of the "expansion operation" we just sketched. Of course, instances that are edge-coloured are easy to convert to instances that are node-coloured by subdividing each edge and assigning its colour to the newly introduced node. The results of these experiments are presented in Table 5.

Here $n = 3\cdot 2^\kappa - 2$ and the colours are either $1,\ldots,2\kappa+1$ or $2,\ldots,2\kappa+2$. There are two variants of the game; the second game "g2" is obtained from the first game "g1" by switching the player and incrementing all colours by one. The essential difference between the two versions is which player is tracked by the basic au-value iteration algorithm and this makes a huge difference; however, the modified au-value iteration algorithm performs on both with only slight differences that are not visible in the above table at all (except for a small difference in the $\kappa = 14$-instance). The invariance against player swapping is expected and stems from the fact that the modified au-value iteration algorithm in either case tracks witnesses from the perspective of both players in parallel.

However, what we do find surprising to see is the very fast runtime of our adapted au-value algorithm and our basic au-value iteration from the perspective of the losing player (the t(bas,g2)-column), for the higher choices of κ. Our Theorem 8 states that the forward au-index exceeds the register index, meaning that under forward evaluation, κ is a lower bound on the number of fields required in our witness for determining that player *Even* is the winner in these examples. If the forward au-index is close to the (global) backward au-index (which is the relevant one for our value iteration algorithms), then that would imply that our value iteration algorithms will have to go through roughly 2^{13} iterations before any vertex is assigned a witness with any colour in its 13th coordinate, which should take much more time that the times stated in the table. A potential explanation of this phenomenon could thus be that in general the backward au-index can be very different from the forward au-index, and that in general it is not true that the register index is a lower bound on the backward au-index (contrary to the relationship we established between the register index and the forward au-index). Other factors that may contribute to the faster runtime is the asymmetry between the players with respect to the lifting operation used in the algorithm: For the "primary" player, the lifting operation is defined by taking a *maximum* over the antagonistic updates of the neighbouring witnesses, while for the "opposing" player, this is a *minimum*. Thus, swapping the roles of the two players may cause the lifting operation to progress faster through the $\sqsupseteq$-order on the witnesses.

One can alternatively regard Lehtinen's construction of games with high register index as a scheme where one expands any given parity game roughly by a factor 2^κ by

Table 4 Experimental results for our basic au-value iteration algorithm running on examples of Sect. 10. Reported times are expressed in seconds

k	Nodes	Time for: k	for $k+1$	for $k+2$	for $k+3$	for $k+4$
16	32	1	2	2	2	3
21	42	4	4	5	6	7
26	52	8	10	12	14	256

Table 5 Experimental results for our value iteration algorithms on the examples formed through the construction described in [26], to form games with arbitrarily high register index

κ	Nodes	Colours	t(bas,g1) (s)	t(bas,g2) (s)	t(mod,g1) (s)	t(mod,g2) (s)
7	382	15/16	1	1	1	1
8	766	17/18	2	1	1	1
9	1534	19/20	9	1	1	1
10	3070	21/22	39	1	1	1
11	6142	23/24	183	1	1	1
12	12286	25/26	842	1	1	1
13	24,574	27/28	3910	1	1	1
14	49,150	29/30	19,351	1	6	6

Table 6 Experimental results on a modified class of examples taken from [26], for both our basic and modified au-value iteration algorithms

κ	Nodes	Colours	Time(bas) (s)	Time(mod) (s)
0	20	20	1	1
1	44	26	2	1
2	92	28	90	3
3	188	30	4227	147
4	380	32	10,000++	7302

In the table, "10,000++" means that the program needed more than 10,000 s to terminate and was aborted

means of κ rounds of the expansion operation, where in each round we connect one of the nodes of the two copies through a bidirected edge with two newly introduced colours. In the case that one does not start with the one-node game as in [26], but instead with one of the lower bound games of Sect. 10, one obtains a series of games that is difficult for both versions of the au-value iteration algorithm. The experimental results on this class of games are displayed in Table 6.

Note that independently of the starting game, if we inspect the sequence of instances resulting from repeated applications of the expansion operation, we observe that the number of colours behaves linearly in κ and logarithmically in the number of nodes of the instance. Thus, the overall performance of all the quasi-polynomial time algorithms discussed in this paper will be captured by some fixed polynomial in the number of nodes of the games of these families of instances. For finding witnesses on which the modified au-value iteration algorithm exhibits quasi-polynomial time behaviour, some more sophisticated choice of the examples would be needed. The authors are confident that this can be done.

12 Discussion

The main contribution of this paper has been to show how to adapt the quasi-polynomial time approach of [6] so that its space complexity becomes polynomial, and in particular quasi-linear. To this end, we adapted the witnesses introduced in [6] in order to have them satisfy certain monotonicity properties that make them suitable for use as a progress measure. This in turn allows for a quasi-linear algorithm by computing a minimal consistent progress measure through a lifting procedure.

A second contribution has been to consider the length of a progress measure as a complexity measure for solving a parity game. Our analysis to this end gave us an adapted algorithm where the time complexity is parametrised by the value of the new complexity measure, and also runs in quasi-polynomial time (and quasi-linear space) in the worst case. We compared our new complexity measure, the au-index, to Lehtinen's register index [26], and provided a proof that our index always exceeds it. This gives as a corollary an alternative proof of the fact that the register index is at most $\lfloor \log n \rfloor + 1$. Our experimental results show that our new algorithms are very efficient on many important classes of parity games and can outperform the current state of the art.

An interesting specific research question that we leave open in this work is whether the au-index with respect to backward (rather than forward) evaluation of a play can in some way be related to its forward version, or to the register index. Also, we would like to see an answer to the question whether the au-index can be upper bounded in terms of the register index, in addition to the current lower bound that we proved in the present work.

Acknowledgements We thank Xiang Fei Ding for pointing out an error in a previous version of this manuscript. We thank Tin Lok Wong for help with programming and implementation of the algorithms. We thank anonymous reviewers for many helpful suggestions and improvements made to a preliminary submission of this manuscript. Sanjay Jain was supported in part by NUS Grant C252-000-087-001. Further, Sanjay Jain and Frank Stephan were supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 Grant MOE2016-T2-1-019 / R146-000-234-112. Sven Schewe and Dominik Wojtczak were supported in part by EPSRC Grant EP/M027287/1. Further, John Fearnley, Bart de Keijzer, Sven Schewe and Dominik Wojtczak were supported in part by EPSRC Grant EP/P020909/1.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (2002)
2. Ash, R.B.: Information Theory. Dover Publications Inc., New York (1990)
3. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: Dag-width and parity games. In: Proceedings of STACS, pp. 524–436. Springer, Berlin (2006)
4. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Appl. Math. **155**(2), 210–229 (2007). https://doi.org/10.1016/j.dam.2006.04.029
5. Browne, A., Clarke, E.M., Jha, S., Long, D.E., Marrero, W.: An improved algorithm for the evaluation of fixpoint expressions. TCS **178**(1–2), 237–255 (1997)
6. Calude, C.S., Jain, S., Khoussainov, B., Li, W., Stephan, F.: Deciding parity games in quasipolynomial time. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 252–263. ACM (2017)
7. Chatterjee, K., Henzinger, M., Loitzenbauer, V.: Improved algorithms for one-pair and k-pair streett objectives. In: Proceedings of LICS, pp. 269–280. IEEE Computer Society (2015)
8. de Alfaro, L., Henzinger, T.A., Majumdar, R.: From verification to control: dynamic programs for omega-regular objectives. In: Proceedings of LICS, pp. 279–290 (2001)
9. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Proceedings of CAV, pp. 385–396 (1993)
10. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proceedings of FOCS, pp. 368–377. IEEE Computer Society Press (1991)
11. Emerson, E.A., Lei, C.: Efficient model checking in fragments of the propositional μ-calculus. In: Proceedings of LICS, pp. 267–278. IEEE Computer Society Press (1986)
12. Fearnley, J., Jain, S., Schewe, S., Stephan, F., Wojtczak, D.: An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10–14, 2017, pp. 112–121. ACM (2017)
13. Fearnley, J.: Non-oblivious strategy improvement. In: Proceedings of LPAR, pp. 212–230 (2010)
14. Friedmann, O., Lange, M.: PGSolver version 4.0 (2017). https://github.com/tcsprojects/pgsolver
15. Friedmann, O., Lange, M.: Solving parity games in practice. In: Proceedings of ATVA, pp. 182–196 (2009)
16. Friedmann, O., Lange, M.: The PGSolver collection of parity game solvers. University of Munich (2014). http://www.win.tue.nl/~timw/downloads/amc2014/pgsolver.pdf
17. Friedmann, O.: An exponential lower bound for the latest deterministic strategy iteration algorithms. LMCS **7**(3) (2011)
18. Friedmann, O.: Recursive algorithm for parity games requires exponential time. RAIRO-Theor. Inf. Appl. **45**(4), 449–457 (2011)
19. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: A simple algorithm for solving qualitative probabilistic parity games. In: Proceedings of CAV, LNCS, vol. 9780, pp. 291–311 (2016)
20. Jurdziński, M., Lazić, R.: Succinct progress measures for solving parity games. In: Proceedings of LICS 2017, p. (to appear) (2017). arXiv:1702.05051
21. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of SODA, pp. 117–123. ACM/SIAM (2006)
22. Jurdziński, M.: Small progress measures for solving parity games. In: Proceedings of STACS, pp. 290–301. Springer, Berlin (2000)
23. Jurdziński, M.: Deciding the winner in parity games is in UP $\cap$ co-UP. Inf. Process. Lett. **68**(3), 119–124 (1998)
24. Kozen, D.: Results on the propositional μ-calculus. TCS **27**, 333–354 (1983)
25. Lange, M.: Solving parity games by a reduction to SAT. In: Proceedings of International Workshop on Games in Design and Verification (2005)
26. Lehtinen, K.: A modal μ perspective on solving parity games in quasi-polynomial time. To appear in the proceedings of LiCS'18
27. Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Inf. Comput. **117**(1), 151–155 (1995)
28. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Log. **65**(2), 149–184 (1993)
29. Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Proceedings of CAV, pp. 80–92. Springer, Berlin (2003)
30. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proceedings of LICS, pp. 255–264. IEEE Computer Society (2006)
31. Puri, A.: Theory of hybrid systems and discrete event systems. Ph.D. thesis, Computer Science Department, University of California, Berkeley (1995)
32. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: Proceedings of LOPSTR, pp. 127–142. Springer, Berlin (2006)
33. Schewe, S., Finkbeiner, B.: The alternating-time μ-calculus and automata over concurrent game structures. In: Proceedings of CSL, pp. 591–605. Springer, Berlin (2006)
34. Schewe, S., Trivedi, A., Varghese, T.: Symmetric strategy improvement. In: Proceedings of ICALP, LNCS, vol. 9135, pp. 388–400 (2015)
35. Schewe, S.: An optimal strategy improvement algorithm for solving parity and payoff games. In: Proceedings of CSL 2008, pp. 368–383. Springer, Berlin (2008)
36. Schewe, S.: Solving parity games in big steps. J. Comput. Syst. Sci. **84**, 243–262 (2017)
37. Totzke, P.: Implementation of the succinct progress measures algorithm from [22] (2017). https://github.com/pazz/pgsolver/tree/sspm
38. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Proceedings of ICALP, pp. 628–641. Springer, Berlin (1998)
39. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Proceedings of the CAV, pp. 202–215. Springer, Berlin (2000)
40. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bull. Soc. Math. Belg. **8**(2), 359 (2001)
41. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**(1–2), 135–183 (1998)