International Journal of Foundations of Computer Science
© World Scientific Publishing Company

THE STEVENS-STIRLING-ALGORITHM FOR SOLVING PARITY GAMES LOCALLY REQUIRES EXPONENTIAL TIME

OLIVER FRIEDMANN
Institut für Informatik, Ludwig-Maximilians-Universität
Oliver.Friedmann@googlemail.com
http://www.tcs.ifi.lmu.de/~friedman

This paper presents a new lower bound for the model-checking algorithm based on solving parity games due to Stevens and Stirling. We outline a family of games of linear size on which the algorithm requires exponential time.

Keywords: parity game; lower bound; mu calculus model checking; stevens-stirling algorithm; exponential bound

1. Introduction

Parity games are simple two-player games of perfect information played on directed graphs whose nodes are labeled with natural numbers, called priorities. A play in a parity game is an infinite sequence of nodes whose winner is determined by the highest priority occurring infinitely often.

Parity games have various applications in computer science, e.g. as algorithmic back end to the model checking problem of the modal μ-calculus [1, 2] or in the theory of formal languages and automata in particular [3, 4].

Solving parity games can be divided into global and local solving: while global solvers determine for each position in the game which player can win starting from there, local solvers are additionally given one single position in the game for which it should be determined who wins starting from there.

Clearly, the local and global problem are interreducible, the global one solves the local one for free, and the global one is solved by calling the local one linearly many times. But neither of these indirect methods is particularly clever. Thus, there are algorithms for the global, and other algorithms for the local problem. We focus on the local problem here which particularly applies for the model checking problem of the modal μ-calculus, as one is only interested in determining who wins the position that corresponds to the initial model checking tuple containing the root formula and the initial state of the transition system.

There are many algorithms that solve parity games globally, but surprisingly there is only one algorithm – at least to our knowledge – that is a genuine local solver, namely the one by Stevens and Stirling [5], to which we will refer to as the *model checking algorithm*. In fact, it is directly defined as a model checking problem in [5]; since μ-calculus model-checking and solving parity games are interreducible, we will study the model checking algorithm directly as a local parity game solving algorithm in this paper.

It basically explores a game depth-first and whenever it reaches a cycle, it stops, storing the node starting the cycle along with a cycle progress measure as an *assumption* for the cycle-winning player. Then, the exploration is backtracked in the sense that if the losing player could have made other moves, they are again explored depth-first. If this leads to a cycle-win for the other player, the whole process starts again, now with respect to the other player. Whenever the backtracking finally leads to the starting node of a cycle, the node is registered as a *decision* for the player which basically can be seen as being a preliminary winning node for the respective player. Additionally, if there are assumptions of the other player for the respective node, these assumptions are dropped, and all depending assumptions and decisions are invalidated.

The rest of the paper is organized as follows. Section 2 defines the basic notions of parity games and some notations that are employed throughout the paper. Section 3 recaps the model checking algorithm. In Section 4, we outline a family of games on which the algorithm requires an exponential number of iterations.

2. Parity Games

A *parity game* is a tuple $G = (V, V_0, V_1, E, \Omega)$ where (V, E) forms a directed graph whose node set is partitioned into $V = V_0 \cup V_1$ with $V_0 \cap V_1 = \emptyset$, and $\Omega : V \to \mathbb{N}$ is the *priority function* that assigns to each node a natural number called the *priority* of the node. We assume the underlying graph to be total, i.e. for every $v \in V$ there is a $w \in W$ s.t. $(v, w) \in E$. In the following we will restrict ourselves to finite parity games.

We also use infix notation vEw instead of $(v, w) \in E$ and define the set of all *successors* of v as $vE := \{w \mid vEw\}$. The size $|G|$ of a parity game $G = (V,\ V_0,\ V_1,\ E,\ \Omega)$ is defined to be the cardinality of E, i.e. $|G| := |E|$; since we assume parity games to be total w.r.t. E, this seems to be a reasonable way to measure the size.

The game is played between two players called 0 and 1: starting in a node $v_0 \in V$, they construct an infinite path through the graph as follows. If the construction so far has yielded a finite sequence $v_0 \dots v_n$ and $v_n \in V_i$ then player i selects a $w \in v_nE$ and the play continues with the sequence $v_0 \dots v_n w$.

Every play has a unique winner given by the *parity* of the greatest priority that occurs infinitely often. The winner of the play $v_0v_1v_2\dots$ is player i iff $\max\{p \mid \forall j \in \mathbb{N}\, \exists k \geq j : \Omega(v_k) = p\} \equiv_2 i$ (where $i \equiv_k j$ holds iff $|i - j| \mod k = 0$). That is, player 0 tries to make an even priority occur infinitely often without any greater odd priorities occurring infinitely often, player 1 attempts the converse.

We will often denote plays by $\pi = v_0v_1v_2\dots$ and identify $\pi(i)$ with the respective

v_i. The *length* of a finite play π is denoted by $|\pi|$ and equals the number of nodes occurring in the play. Particularly, $\pi(|\pi|-1)$ denotes the last node of a finite play π.

A *strategy* for player i is a – possibly partial – function $\sigma : V^*V_i \to V$, s.t. for all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_jE$ for all $j = 0, \dots, n-1$, and all $v_n \in V_i$ with $v_0 \dots v_n \in \mathtt{dom}(\sigma)$ we have: $\sigma(v_0 \dots v_n) \in v_nE$. A play $v_0v_1 \dots$ *conforms* to a strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ and $v_0v_1 \dots v_j \in \mathtt{dom}(\sigma)$ then $v_{j+1} = \sigma(v_0 \dots v_j)$.

Intuitively, conforming to a strategy means to always make those choices that are prescribed by the strategy. A player i strategy σ is *complete* w.r.t. a node v iff for every play $v_0v_1 \dots$ with $v_0 = v$ conforming to σ and every j with $v_j \in V_i$ holds: $v_0v_1 \dots v_j \in \mathtt{dom}(\sigma)$.

A strategy σ for player i is a *winning strategy* in node v if σ is complete w.r.t. v and player i wins every play that begins in v and conforms to σ. We say that player i *wins* the game G starting in v iff he or she has a winning strategy for G starting in v.

A strategy σ for player i is called *positional* if for all $v_0 \dots v_n \in V^*V_i \cap \mathtt{dom}(\sigma)$ and all $w_0 \dots w_m \in V^*V_i$ we have: if $v_n = w_m$ then $w_0 \dots w_m \in \mathtt{dom}(\sigma)$ and $\sigma(v_0 \dots v_n) = \sigma(w_0 \dots w_m)$. That is, the value of the strategy on a finite path only depends on the last node on that path.

With G we associate two sets $W_0, W_1 \subseteq V$ such that W_i is the set of all nodes v s.t. player i wins the game G starting in v. Here we restrict ourselves to positional strategies because it is well-known that parity games enjoy positional determinacy meaning that for every node v in the game either $v \in W_0$ or $v \in W_1$ [4] with corresponding positional winning strategies.

The problem of *solving* a parity game is, roughly speaking, to determine which of the players has a winning strategy for that game. However, this does not take starting nodes into account. In order to obtain a well-defined problem, this is refined in two ways.

The problem of solving the parity game G *globally* is to determine, for *every* node $v \in V$ whether $v \in W_0$ or $v \in W_1$. The problem of solving G *locally* is to determine for a *given* $v \in V$ whether $v \in W_0$ or $v \in W_1$ holds. In addition, a local solver should return one strategy that is a winning strategy for player i if $v \in W_i$ for the given input node v. A global solver should return two strategies, one for each player s.t. player i wins exactly on the nodes $v \in W_i$ if he/she plays according to the strategy computed for her.

3. The model checking algorithm

The model checking algorithm by Stevens and Stirling basically explores the game, starting in the initial node, depth-first until it encounters a repeat. It relies on interrelated data structures, called *decisions* and *assumptions*, that have to be organized in a dependency ordering. Instead of managing a whole dependency graph, the authors pursue a simplified approach relying on time-stamping.

As a drawback, this simplified approach may lead to a removal of more decisions than necessary in a run of the algorithm; on the other hand, it seems to be faster in practice [5] and remains sound and complete. We note that our lower bound construction does not rely on the mechanism of dependency since it never happens that decisions have to be invalidated.

In order to compare the profitability of certain plays in the explored game, the algorithm introduces a structure called *index*. Essentially, the index of a play π denotes for every occurring priority p how often it occurs in π without seeing any greater priorities afterwards.

Let G be a parity game. A *G-index* is a map $i : \mathtt{ran}(\Omega_G) \to \mathbb{N}$; let π be a finite play. The *π-associated index* i_π is defined by

$$i_\pi : p \mapsto |\{j < |\pi| \mid \Omega(\pi(j)) = p \text{ and } \Omega(\pi(k)) \leq p \text{ for every } k > j\}|$$

Let $\mathbf{0}$ denote the index that maps every priority to 0. The index of a play can be calculated inductively by applying the following *priority addition function* $i \oplus p$, which takes an index i and a priority p and is defined as follows

$$(i \oplus p)(q) := \begin{cases} i(q) & \text{if } q > p \\ i(q) + 1 & \text{if } q = p \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see that $i_\pi = (\ldots(\mathbf{0} \oplus \Omega(\pi(0))) \oplus \ldots) \oplus \Omega(\pi(|\pi| - 1))$.

Next, we define a total ordering w.r.t. a given player $u \in \{0, 1\}$ on indices that intuitively measures the usefulness of indices w.r.t. player u. For two indices i and j let $i >_u j$ hold iff there is some $p \in \mathtt{ran}(\Omega_G)$ s.t.

- $i(p) \neq j(p)$,
- $i(h) = j(h)$ for all $h > p$ and
- $i(p) > j(p)$ iff $p \equiv_2 u$

This p will be called the *most significant difference between i and j*. In other words, $i >_u j$ is a lexicographic ordering on indices based on the u-reward ordering on the components of the indices. In order to denote that $i >_u j$ holds with p being the most significant difference, we also write $i >_u^p j$.

By considering the lexicographic ordering on indices induced by the relevance ordering on components, it is not too hard to see that the following holds:

Corollary 1. *Let G be a parity game, π and π' be plays in G with $|\pi| < |\pi'|$ and $\pi(k) = \pi'(k)$ for all $k < |\pi|$. Then $i_\pi \neq i_{\pi'}$, hence either $i_\pi >_0 i_{\pi'}$ or $i_\pi >_1 i_{\pi'}$.*

Again, it is easy to see that the most significant differences between i and j, and j and k can be used to obtain the most significant difference between i and k.

Corollary 2. *Let G be a parity game, i, j and k be G-indices, $p \in \{0, 1\}$ be a player and w and q be priorities.*

(1) If $i <_p^w j$, $i <_p^q k$ and $q < w$ it follows that $k <_p^w j$.
(2) If $j <_p^w i$, $k <_p^q i$ and $q < w$ it follows that $j <_p^w k$.
(3) If $i <_p^w j$ and $j <_p^q k$ it follows that $i <_p^{\max(w,q)} k$.

The intuition behind an index is similar to the idea of the discrete valuations in the strategy iteration: giving an abstract description of an associated play disregarding the ordering. While valuations in strategy iteration essentially include all relevant nodes in the play without regarding the ordering, an index in contrast is a window to the immediate past of a play, in the sense that smaller priorities are hidden before the last occurrence of a higher priority.

Exploring the game, the algorithm maintains a playlist storing for each node in the play the index associated with the node, which edges originating from the node remained unvisited, the time at which it was added to the playlist and if it was used as an *assumption* for one or both of the two players.

Formally, a *playlist entry* is a tuple (v, i, t, b, a) with $v \in V_G$, i a G-index, $t \subseteq vE_G$, $b \in \mathbb{N}$ and $a \subseteq \{0, 1\}$. A *playlist* is a map l with a domain $\{0, \ldots, k-1\}$ for some $k \in \mathbb{N}$ that maps each $i < k$ to a playlist entry. The length of a playlist l is denoted by $|l| := k$ and the empty playlist is denoted by $[]$. To access the i-th entry of the playlist, we write l_i.

Let l be a playlist and e be a playlist entry. Adding e to the top of l is denoted by $e :: l$ and formally defined as follows: $e :: l$ is the playlist with the domain $\{0, \ldots, |l|\}$ that maps every $i < |l|$ to l_i and $i = |l|$ to e.

When comparing two playlists l and l', we are often not interested in whether they are differing in the assumption component of one playlist entry. Hence, we define $l \equiv l'$ to hold iff $|l| = |l'|$ and for every $k < |l|$, we have that $l_k = (v, i, t, b, \_)$ implies $l'_k = (v, i, t, b, \_)$ (an occurrence of $\_$ in a tuple is to be seen as an unbound existentially quantified variable).

A *decision* for a player p at a node v is a triple (i, t, u) where i is a G-index, $t \in \mathbb{N}$ is a time-stamp and $u \in V_G \cup \{\bot\}$ s.t. $u = \bot$ if $v \notin V_p$ and $u \in vE_G$ if $v \in V_p$. Intuitively, the decision (i, t, u) at v for p tells us that if a play reaches v with an index j that is not p-worse than i, it can be won by p if all assumptions before t actually hold true. Additionally, if v is a p-choice point, u denotes the corresponding strategy decision that should be played.

During a run of the algorithm, it happens that decisions are removed, depending on their time-stamp. For instance, if a node v was used as an assumption for p at a time t and it turns out later on that v will now be used as a decision for $1-p$, all decisions for p that have been made after t are to be removed. Hence, the algorithm maintains a set of decisions for each player and each node.

Formally, a *decision map* for player p is a map d with domain V_p that maps each node $v \in V_p$ to a set of decisions for p at v. The decision map assigning to each node the empty set is denoted by $\mathbf{e}$. A decision map d for player p induces a p-strategy σ_d which is defined on each node v with $d(v) \neq \emptyset$ as follows: $\sigma_d(v) = u$ iff there is a time t s.t. $(\_, t, u) \in d(v)$ and for all $(\_, s, \_) \in d(v)$ it holds that $s \leq t$.

Whenever the algorithm hits a repeat while exploring the game, the player p winning the cycle is determined, and the starting node of the cycle in the playlist is marked to be used as an assumption for p. Then, the playlist is backtracked in order to determine whether player $1-p$ could have made better choices. Additionally, if exploring the game reaches a node at which a decision d for either one of the players p is *applicable*, meaning that the current index is p-greater than the decision index, the backtracking process is also invoked. Explore accepts six parameters: the game G, the current node v, the current index i, the current playlist l, the time-counter c and the tuple of decision stacks d for both players. See Algorithm 1 for a pseudo-code specification.

Algorithm 1 model checking algorithm: Explore Routine

```
1: procedure Explore($G$, $v$, $i$, $l$, $c$, $d$)
2:     if $(\exists p \in \{0,1\}. \exists(j, \_, \_) \in d_p(v) : i \geq_p j)$ then
3:         return Backtrack($G$, $v$, $p$, $l$, $c+1$, $d$)
4:     else if $(\exists i < |l| : l_i = (v, j, t, b, a))$ then
5:         $p \leftarrow \{0,1\}$ s.t. $i >_p j$ [a]
6:         return Backtrack($G$, $v$, $p$, $l[i \mapsto (v, j, t, b, a \cup \{p\})]$, $c+1$, $d$)
7:     else
8:         $w \leftarrow$ Select($G$, $v$, $vE$)
9:         return Explore($G$, $w$, $i \oplus \Omega(w)$, $(v, i, vE \setminus \{w\}, c, \emptyset) :: l$, $c+1$, $d$)
10:    end if
11: end procedure
```

The algorithm leaves the choice of selecting the next successor, that is to be visited, open: for a game G, the function Select chooses for every node v and for every non-empty successor subset $\emptyset \neq t \subseteq vE_G$, a node $\text{Select}(G, w, t) \in t$.

Backtracking passes through the playlist in reverse order until it finds a choice point of the other player that still contains unexplored edges. While backtracking, all nodes that are removed from the top of the playlist are added to the decision set of the player for whom the playlist is backtracked. Whenever adding a decision for a player p at v, the algorithm checks whether v was used as an assumption for $1-p$ and if so, all depending $1-p$ decisions are removed.

If the backtracking processes finally encounters a choice point of the other player with unexplored choices, the exploration process continues at that node. Otherwise, the algorithm finishes. Backtrack accepts six parameters: the game G, the current node v, the player p for which backtracking is to be performed, the playlist l, the time-counter c and the tuple of decision stacks d for both players. See Algorithm 2 for a pseudo-code specification.

The model checking algorithm is then realized by the Decide routine that takes

[a]Holds for either one of the two players due to Corollary 1.

Algorithm 2 model checking algorithm: Backtrack Routine

```
procedure BACKTRACK($G$, $v$, $p$, $l$, $c$, $(d_0, d_1)$)
    if $(l = [])$ then
        return $(p, \sigma_{d_p})$
    else
        $(w, i, t, b, a) :: m \leftarrow l$
        if $(w \in V_p)$ or $(t = \emptyset)$ then
            $u \leftarrow \begin{cases} \bot & \text{if } t = \emptyset \\ v & \text{otherwise} \end{cases}$
            $d'_p \leftarrow d_p[w \mapsto d_p(w) \cup \{(i, c, u)\}]$
            $d'_{1-p} \leftarrow \begin{cases} y \mapsto \{(j, s, z) \in d_{1-p}(y) \mid j < b\} & \text{if } (1-p) \in a \\ d_{1-p} & \text{otherwise} \end{cases}$
            return BACKTRACK($G$, $w$, $p$, $m$, $c$, $(d'_0, d'_1)$)
        else
            $u \leftarrow$ SELECT($G$, $w$, $t$)
            $l' \leftarrow (w, i, t \setminus \{u\}, b, a) :: m$
            return EXPLORE($G$, $u$, $i \oplus \Omega(u)$, $l'$, $c$, $(d_0, d_1)$)
        end if
    end if
end procedure
```

a game G and node v for which the winner is to be decided along with a winning strategy. It just invokes the EXPLORE-routine by initializing all required parameters with the default values. See Algorithm 3 for a pseudo-code specification.

Algorithm 3 model checking algorithm: Decide Routine

```
procedure DECIDE($G$, $v$)
    return EXPLORE($G$, $v$, $\mathbf{0} \oplus \Omega(v)$, $[]$, 1, $(\mathbf{e}, \mathbf{e})$)
end procedure
```

It is easy to see that the model checking algorithm always terminates. Correctness essentially follows from the observation that the algorithm eventually explores a winning strategy for one of the two players and every backtracking operation results in indices that are won by the respective player.

Theorem 3 ([5]) *Let G be a parity game and v be a node in G. Calling* DECIDE(G, v) *terminates and returns a tuple (p, σ) s.t. $v \in W_p$ and σ is a p-winning strategy starting in v.*

For the analysis of the runtime complexity, let $\texttt{MC}(G, v)$ denote the total number of EXPLORE-calls that are executed in order to solve a given parity game G and initial node v.

A trivial upper bound can be easily derived: due to the fact the algorithm explores every path ending in a cycle at most once, it follows that the depth of the search tree is bounded by the number of nodes and the out-degree of every point in the search tree is bounded by the out-degree of respective node in the game. Hence, a trivial upper bound on the runtime complexity is $\mathcal{O}(n^n)$ assuming that n is the number of nodes in the game.

Theorem 4. *Let G be a parity game and $v \in G$. Then $\texttt{MC}(G, v) \in 2^{\mathcal{O}(n \cdot \log n)}$.*

4. Exponential Lower Bound

We present a concrete family of parity games on which the expected runtime of the model checking algorithm is exponential. Obviously, the analysis of the lower bound depends to some extent on the selection policy Select. We follow the most natural selection policy here that selects a successor node uniformly at random.

$$\textsc{Select}^R(G, v, R) \equiv u \in R \text{ with probability } \frac{1}{|R|}$$

Employing this selection policy, we prove that the expected number of steps is at least exponential on a certain family of games. Nevertheless, for other reasonable policies, it is possible to show that a lower bound on the expected number of steps is also exponential in a similar way.

The games will be denoted by $G_n = (V_n, V_{n,0}, V_{n,1}, E_n, \Omega_n)$ and are of linear size. All nodes are owned by player 1.

$$V_n := \{a_0, \dots, a_n, b_1, \dots, b_n, c_1, \dots, c_n\}$$

The priorities and edges are described in Table 1. The game G_2 is depicted in Figure 1.

Node	Priority	Successors
a_0	0	$\{a_n\}$
$a_{i>0}$	$2 \cdot i$	$\{b_i\}$
b_i	0	$\{c_i, a_{i-1}\}$
c_i	$2 \cdot i - 1$	$\{a_{i-1}\}$

Table 1. The Model Checking Lower Bound Game G_n

Fact 5. *The game G_n has $3 \cdot n + 1$ nodes, $4 \cdot n + 1$ edges and $2 \cdot n$ as highest priority. In particular, $|G_n| = \mathcal{O}(n)$.*

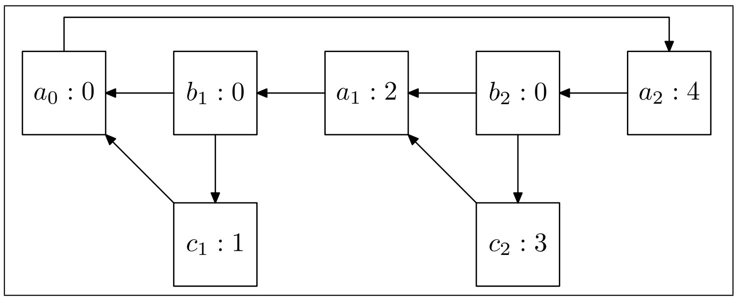

Fig. 1. The Model Checking Lower Bound Game G_3

Obviously, G_n is completely won by player 0, because every cycle eventually goes through a_n, which has the highest priority in the game and is even. The exponential behavior on these games is enforced as follows: assume for the time being that the selection policy always select a_{i-1} first and then c_i.

Exploring the game starting in a_n, assume that the process is currently at b_i choosing to advance to a_{i-1}. Eventually, a_n will be reached again and the play will be backtracked w.r.t. player 0. The backtracking process finally moves back to b_i, advancing to the unexplored c_i subsequently.

Now note that c_i has an odd priority that is greater than all other priorities occurring afterwards, i.e. $\Omega(c_i) > \Omega(q_j)$ with $j < i$ and $q_j \in \{a_j, b_j, c_j\}$. Hence, there is no applicable decision that has been added during the backtracking process. Therefore, advancing from c_i again to a_{i-1} recursively, starts the whole process again.

Otherwise, if the selection policy chooses c_i first instead of a_{i-1}, there will be an applicable decision afterwards, avoiding the second recursive descent.

In order to prove the described behavior of the algorithm correct, we define three predicates $\Psi_n(j, i, l, d)$, $\Phi_n(j, i, d)$ and $\Delta_n(j, i, d)$ that will be used as pre- and postconditions in the induction. Let $j \leq n$, i be an G_n-index, l be a G_n-playlist and $d = (d_0, d_1)$ be G_n-decisions.

- $\Psi_n(j, i, l, d)$ is defined to hold iff all of the following conditions hold:
  (a) $l_k = (q, \_, \_, \_, \_)$ implies that there is an $h > j$ s.t. $(q \in \{a_h, b_h, c_h\})$ for every $k < |l|$
  (b) If $j < n$: $l_0 = (a_n, \mathbf{0} \oplus \Omega(a_n), 0, \emptyset, \_)$
  (c) $i(\Omega(a_n)) = 1$
  (d) $d_1(q) = \emptyset$ for all $q \in V_n$
- $\Phi_n(j, i, d)$ is defined to hold iff the following condition holds:
  (a) $(k, \_, \_) \in d_0(q)$ implies that there is a w s.t. $i <_0^w k$ with $\Omega(w) > 2j$ for all $q \in V_n$ and all G_n-indices k
- $\Delta_n(j, i, d)$ is defined to hold iff all of the following conditions hold:

(a) $(k, \_, \_) \in d_0(q)$ with $i >_0 k$ implies that there is a w s.t. $i >_0^w k$ with $\Omega(w) < 2j$ for all $q \in V_n$ and all G_n-indices k
(b) $(i, \_, \_) \in d_0(a_j)$

Again, we analyze the runtime complexity of the model checking algorithm in terms of $\mathtt{MC}(G, v)$, i.e. the time-counter that is maintained by the EXPLORE-routine. The following function f_n will be shown to capture the progression of it accurately. Let $n \in \mathbb{N}$ and $i < n$.

$$f_n : i \mapsto \begin{cases} 1 & \text{if } i = 0 \\ f_n(i-1) + 4 & \text{if } i > 0 \text{ and } \textsc{Select}(G_n, b_i, b_i E) = c_i \\ 2 \cdot f_n(i-1) + 4 & \text{otherwise} \end{cases}$$

Lemma 6. *Let $j \leq n$, i be an G_n-index, l be a G_n-playlist, $d = (d_0, d_1)$ be G_n-decisions and $c \in \mathbb{N}$ s.t. $\Psi_n(j, i, l, d)$ and $\Phi_n(j, i, d)$ hold.*

Calling EXPLORE(G_n, a_j, i, l, c, d) *leads to* BACKTRACK$(G_n, a_j, 0, l', c', d')$ *where $c' = c + f_n(j) + 1$ and $l \equiv l'$ s.t. $\Psi_n(j, i, l', d')$ and $\Delta_n(j, i, d')$ hold.*

Proof. By induction on j.

For $j = 0$, let $\Psi_n(0, i, l, d)$ and $\Phi_n(0, i, d)$ hold; EXPLORE(G_n, a_0, i, l, c, d) directly invokes EXPLORE$(G_n, a_n, i_1, l_1, c{+}1, d)$ where $i_1 = i \oplus \Omega(a_n)$ and $l_1 = (a_0, i, \emptyset, c, \emptyset) :: l$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a)) nor a repeat in the playlist (Ψ_n (a)).

Running EXPLORE$(G_n, a_n, i_1, l_1, c + 1, d)$ encounters a repeat, as $(a_n, \mathbf{0} \oplus \Omega(a_n), 1, \emptyset, \_)$ is in l_1 by Ψ_n (b). Due to the fact that $i(\Omega(a_n)) = 1$ by Ψ_n (c) and $i_1(\Omega(a_n)) = 2$, it directly follows that the repeat is profitable for player 0.

Hence, BACKTRACK$(G_n, a_n, 0, l_2, c + 2, d)$ is called, where $l_2 \equiv l_1$. Since l_2 is not empty and the top entry has no other transitions to visit, BACKTRACK$(G_n, a_0, 0, l', c + 1, d')$ is invoked, where $l' \equiv l$, $d'_1 = d_1$ and $d'_0 = d_0[a_0 \mapsto d_0(a_0) \cup \{(i, c, \bot)\}]$. Note that $\Psi_n(j, i, l', d')$ as well as $\Delta_n(j, i, d')$ hold.

For $j \rightsquigarrow j + 1$, let $\Psi_n(j + 1, i, l, d)$ and $\Phi_n(j + 1, i, d)$ hold; EXPLORE$(G_n, a_{j+1}, i, l, c, d)$ directly invokes EXPLORE$(G_n, b_{j+1}, i_1, l_1, c + 1, d)$ where $i_1 = i \oplus \Omega(b_{j+1})$ and $l_1 = (a_{j+1}, i, \emptyset, c, \emptyset) :: l$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a)) nor a repeat in the playlist (Ψ_n (a)).

Let $w =$SELECT$(G_n, b_{j+1}, b_{j+1}E)$. We will now distinguish on whether $w = c_{j+1}$ or $w = a_j$.

- Case $w = a_j$: Calling EXPLORE$(G_n, b_{j+1}, i_1, l_1, c + 1, d)$ directly invokes EXPLORE$(G_n,a_j,i_2,l_2,c + 2,d)$ where $i_2 = i_1 \oplus \Omega(a_j)$ and $l_2 =(b_{j+1}, i_1, \{c_{j+1}\}, c + 1, \emptyset){::}\ l_1$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 2) nor a repeat in the playlist (Ψ_n (a)).

  Now note that $\Psi_n(j, i_2, l_2, d)$ as well as $\Phi_n(j, i_2, d)$ hold: $\Psi_n(j, \ldots)$ (a) holds by construction of l_2 and $\Psi_n(j + 1, \ldots)$ (a); if $j + 1 < n$, $\Psi_n(j, \ldots)$ (b) holds due to $\Psi_n(j + 1, \ldots)$ (b), otherwise by construction of l_2; $\Psi_n(j, \ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j+1,\ldots)$ (c) and (d). Lastly, $\Phi_n(j,i_2,d)$ holds due to $\Phi_n(j+1,i,d)$ and Corollary 2.

By induction hypothesis, eventually $\textsc{Backtrack}(G_n,a_j,0,l_3,c+f_n(j)+3,d^i)$ gets called, where $l_3 \equiv l_2$ and $\Psi_n(j,i_2,l_3,d^i)$ and $\Delta_n(j,i_2,d^i)$ hold. Since c_{j+1} remained unexplored, $\textsc{Explore}(G_n,c_{j+1},i_4,l_4,c+f_n(j)+3,d^i)$ is invoked, where $i_4 = i_1 \oplus \Omega(c_{j+1})$ and $l_4 \equiv (b_{j+1},i_1,\emptyset,c+1,\emptyset) :: l_1$.

Consider that $\Phi_n(j+1,i_4,d^i)$ holds: let $(k,\_,\_) \in d_0^i(q)$ for an arbitrary q. By construction, $i_4 <_0^{c_{j+1}} i_2$. Hence, if $i_2 \leq_0 k$, it follows by Corollary 2 that $\Phi_n(j+1,i_4,d^i)$ holds. Otherwise, we apply $\Delta_n(j,i_2,d^i)$ and conclude that $i_2 >_0^w k$ with $\Omega(w) < 2j$, thus by Corollary 2 it follows that $\Phi_n(j+1,i_4,d^i)$ holds.

Subsequently, $\textsc{Explore}(G_n,a_j,i_5,l_5,c+f_n(j)+4,d^i)$ gets called, where $i_5 = i_4 \oplus \Omega(a_j)$ and $l_5 \equiv (c_{j+1},i_4,\emptyset,c+f_n(j)+1,d^i) :: l_4$, as there is neither an applicable decision (Ψ_n (d) and $\Phi_n(j+1,i_4,d^i)$ (a)) nor a repeat in the playlist (Ψ_n (a)).

Note that $\Psi_n(j,i_5,l_5,d^i)$ as well as $\Phi_n(j,i_5,d^i)$ hold: $\Psi_n(j,\ldots)$ (a) holds by construction of l_5 and $\Psi_n(j+1,\ldots)$ (a); if $j+1<n$, $\Psi_n(j,\ldots)$ (b) holds due to $\Psi_n(j+1,\ldots)$ (b), otherwise by construction of l_5; $\Psi_n(j,\ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j+1,\ldots)$ (c) and (d). Lastly, $\Phi_n(j,i_5,d^i)$ holds due to $\Phi_n(j+1,i_4,d^i)$ and Corollary 2.

By induction hypothesis, eventually $\textsc{Backtrack}(G_n,a_j,0,l_6,c',d^{ii})$ gets called, where $l_6 \equiv l_5$, $c' = c+2f_n(j)+5 = c+f_n(j+1)+1$ and $\Psi_n(j,i_5,l_6,d^{ii})$ and $\Delta_n(j,i_5,d^{ii})$ hold. Since l_6 is not empty and the top entry has no other transitions to visit, $\textsc{Backtrack}(G_n,c_{j+1},0,l_7,c',d^{iii})$ is invoked, where $l_7 \equiv l_4$ and $d^{iii} = d^{ii}[c_{j+1} \mapsto d^{ii}(c_{j+1}) \cup \{(i_4,c+f_n(j)+1,\bot)\}]$.

Again since l_7 is not empty and the top entry has no other transitions to visit, $\textsc{Backtrack}(G_n,b_{j+1},0,l_8,c',d^{iv})$ is invoked, where $l_8 \equiv l_1$ and $d^{iv} = d^{iii}[b_{j+1} \mapsto d^{iii}(b_{j+1}) \cup \{(i_1,c+1,\bot)\}]$. Lastly, $\textsc{Backtrack}(G_n,a_{j+1},0,l_9,c',d^v)$ is called for the same reasons, where $l_9 \equiv l$ and $d^v = d^{iv}[a_{j+1} \mapsto d^{iv}(a_{j+1}) \cup \{(i,c,\bot)\}]$.

It remains to show that $\Psi_n(j+1,i,l_9,d^v)$ and $\Delta_n(j+1,i,d^v)$ hold: Since $d_2^v = d_2$ and $l_9 \equiv l$, it directly follows that $\Psi_n(j+1,i,l_9,d^v)$ by assuming $\Psi_n(j+1,i,l,d)$. $\Delta_n(j+1,i,d^v)$ (b) obviously holds as $\{(i,c,\bot)\} \in d^v(a_{j+1})$; for $\Delta_n(j+1,i,d^v)$ (a) let $q \in V_n$ be arbitrary s.t. there is $(k,\_,\_) \in d_0^v(q)$ with $i >_0 k$. If $(k,\_,\_) \in d_0^{ii}(q)$ it follows by induction hypothesis that $i_5 >_0^w k$ with $\Omega(w) < 2j$; since $i >_0^{c_{j+1}} i_5$ and $\Omega(c_{j+1}) = 2(j+1)-1$, Corollary 2 implies that $i >_0^{c_{j+1}} k$ with $\Omega(c_{j+1}) < 2(j+1)$. Otherwise, if $(k,\_,\_) \notin d_0^{ii}(q)$, then $k \in \{i,i_1,i_4\}$; only $i_4 <_0 i$, again with c_{j+1}: $i_4 <_0^{c_{j+1}} i$.

- Case $w = c_{j+1}$: Calling $\textsc{Explore}(G_n,b_{j+1},i_1,l_1,c+1,d)$ invokes $\textsc{Explore}(G_n,c_j,i_2,l_2,c+2,d)$ where $i_2 = i_1 \oplus \Omega(c_{j+1})$ and $l_2 = (b_{j+1},i_1,\{a_j\},c+1,\emptyset) :: l_1$, since there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 2) nor a repeat in the playlist (Ψ_n (a)).

Subsequently, $\textsc{Explore}(G_n, a_j, i_3, l_3, c+3, d^i)$ gets called, where $i_3 = i_2 \oplus \Omega(a_j)$ and $l_3 \equiv (c_{j+1}, i_2, \emptyset, c{+}2, d) :: l_2$, as there is neither an applicable decision (Ψ_n (d) and Φ_n (a) by Corollary 2) nor a repeat in the playlist (Ψ_n (a)).

Now note that $\Psi_n(j, i_3, l_3, d)$ as well as $\Phi_n(j, i_3, d)$ hold: $\Psi_n(j, \ldots)$ (a) holds by construction of l_3 and $\Psi_n(j+1, \ldots)$ (a); if $j+1 < n$, $\Psi_n(j, \ldots)$ (b) holds due to $\Psi_n(j+1, \ldots)$ (b), otherwise by construction of l_3; $\Psi_n(j, \ldots)$ (c) and (d) obviously hold by construction and $\Psi_n(j{+}1, \ldots)$ (c) and (d). Lastly, $\Phi_n(j, i_3, d)$ holds due to $\Phi_n(j+1, i, d)$ and Corollary 2.

By induction hypothesis, eventually $\textsc{Backtrack}(G_n, a_j, 0, l_4, c + f_n(j) + 4, d^i)$ gets called, where $l_4 \equiv l_3$ and $\Psi_n(j, i_3, l_4, d^i)$ and $\Delta_n(j, i_3, d^i)$ hold. Since l_4 is not empty and the top entry has no other transitions to visit, $\textsc{Backtrack}(G_n,c_{j+1},0,l_5,c + f_n(j) + 4,d^{ii})$ is invoked, where $l_5 \equiv l_2$ and $d^{ii} = d^i[c_{j+1} \mapsto d^i(c_{j+1}) \cup \{(i_2, c+2, \perp)\}]$.

Since $b_{j+1}Ea_j$ remained unexplored, $\textsc{Explore}(G_n, a_j, i_6, l_6, c{+}f_n(j){+}4, d^{ii})$ is invoked, where $i_6 = i_1 \oplus \Omega(a_j)$ and $l_6 \equiv (b_{j+1}, i_1, \emptyset, c+1, \emptyset) :: l_1$. As $\Delta_n(j, i_3, d^i)$ (b) holds by induction hypothesis, $(i_3, \_, \_) \in d_0^{ii}(a_j)$ and $i_6 >_0 i_3$, a decision is applicable, hence $\textsc{Backtrack}(G_n, a_j, 0, l_6, c', d^i i)$ is invoked where $c' = c + f_n(j) + 5 = c + f_n(j+1) + 1$.

Because l_6 is not empty and the top entry has no other transitions to visit, $\textsc{Backtrack}(G_n, b_{j+1}, 0, l_7, c{+}f_n(j){+}5, d^{iii})$ is invoked, where $l_7 \equiv l_1$ and $d^{iii} = d^{ii}[b_{j+1} \mapsto d^{ii}(b_{j+1}) \cup \{(i_1, c{+}1, \perp)\}]$. Lastly, $\textsc{Backtrack}(G_n, a_{j+1}, 0, l_8, c', d^{iv})$ is called for the same reasons, where $l_8 \equiv l$ and $d^{iv} = d^{iii}[a_{j+1} \mapsto d^{iii}(a_{j+1}) \cup \{(i, c, \perp)\}]$.

It remains to show that $\Psi_n(j+1, i, l_8, d^{iv})$ and $\Delta_n(j+1, i, d^{iv})$ hold: Since $d_2^{iv} = d_2$ and $l_8 \equiv l$, it directly follows that $\Psi_n(j+1, i, l_8, d^{iv})$ by assuming $\Psi_n(j+1, i, l, d)$. $\Delta_n(j+1, i, d^{iv})$ (b) obviously holds as $\{(i, c, \perp)\} \in d^{iv}(a_{j+1})$; for $\Delta_n(j{+}1, i, d^{iv})$ (a) let $q \in V_n$ be arbitrary s.t. there is $(k, \_, \_) \in d_0^{iv}(q)$ with $i >_0 k$. If $(k, \_, \_) \in d_0^i(q)$ it follows by induction hypothesis that $i_3 >_0^w k$ with $\Omega(w) < 2j$; since $i >_0^{c_{j+1}} i_3$ and $\Omega(c_{j+1}) = 2(j+1) - 1$, Corollary 2 implies that $i >_0^{c_{j+1}} k$ with $\Omega(c_{j+1}) < 2(j{+}1)$. Otherwise, if $(k, \_, \_) \notin d_0^i(q)$, then $k \in \{i, i_1\}$; none of them is $<_0$-less than i. □

Lemma 7. *Let $n \in \mathbb{N}$. Then* $\mathtt{MC}(G_n, a_n) = f_n(n) + 1$.

Proof. Calling $\textsc{Decide}(G_n, a_n)$ directly invokes $\textsc{Explore}(G_n,\ a_n,\ \mathbf{0} \oplus \Omega(a_n),\ [],$ $0,\ (\mathbf{e}, \mathbf{e}))$. Note that the given arguments trivially satisfy Ψ_n as well as Φ_n, hence Lemma 6 implies that eventually $\textsc{Backtrack}(G_n, a_n, 0, [], f_n(n) + 1, d')$ with some decisions d' is called. By definition of $\textsc{Backtrack}$, it follows that the algorithm directly terminates, hence it requires $f_n(n) + 1$ $\textsc{Explore}$-steps in total. □

Theorem 8. *Deciding (G_n, a_n) via $\textsc{Decide}(G_n, a_n)$ with $\textsc{Select}^R$ requires an expected number of $9 \cdot 1.5^n - 7$ $\textsc{Explore}$-steps. Particularly, a lower bound on the expected worst-case runtime of the parity game model checking algorithm is $1.5^{\Omega(n)}$.*

Proof. By Lemma 7, it requires $f_n(n)+1$ explore-steps to decide a_n. Since $\textsc{Select}^R$ uniformly selects edges, the expected number of steps $\bar{f}_n$ can be written as follows (with $\bar{f}_n(0) = 1$):

$$\bar{f}_n(i+1) = \frac{1}{2} \cdot (1 \cdot \bar{f}_n(i) + 4) + \frac{1}{2} \cdot (2 \cdot \bar{f}_n(i) + 4) = \frac{3}{2} \cdot \bar{f}_n(i) + 4$$

By induction on $i < n$, it directly follows that $\bar{f}_n(i) = 9 \cdot 1.5^i - 8$, hence particularly $\bar{f}_n(n) + 1 = 9 \cdot 1.5^n - 7$. □

5. Future Work

We have shown that the model checking algorithm has exponential worst-case runtime complexity. Although there is currently no algorithm known that solves parity games in polynomial time, there are global algorithms that perform very well in practice [6], some of them being structural candidates to give rise to a polynomial time algorithm, e.g. the strategy improvement algorithms [7, 8, 9].

For future research, it would be interesting to see whether one of the global algorithms that performs well in practice can be turned into a local algorithm, particularly meaning that the algorithm only explores parts of the game that are necessary to decide which player wins the given initial node.

We believe that the strategy iteration could be used to create a well-performing local algorithm: exploring the game depth-first for one of the players p s.t. the algorithm selects single edges whenever reaching a position owned by p and all edges whenever reaching a position of the opponent yields a proper subgame that can be evaluated using the strategy iteration. If this part of the game is not enough to decide who wins, other choices of player p are pursued resulting in more nodes that are added to the subgame which can be evaluated again using the given valuation from the smaller subgame. This process should be intertwined for both players.

References

[1] E. Emerson, C. Jutla, and A. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th Conf. on CAV, CAV'93*, volume 697 of *LNCS*, pages 385–396. Springer, 1993.

[2] C. Stirling. Local model checking games. In *Proc. 6th Conf. on Concurrency Theory, CONCUR'95*, volume 962 of *LNCS*, pages 1–11. Springer, 1995.

[3] E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games*, LNCS. Springer, 2002.

[4] E. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, 1991. IEEE.

[5] P. Stevens and C. Stirling. Practical model-checking using games. In B. Steffen, editor, *Proc. 4th Int. Conf. on Tools and Alg. for the Constr. and Analysis of Systems, TACAS'98*, volume 1384 of *LNCS*, pages 85–101. Springer, 1998.

[6] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *ATVA*, pages 182–196, 2009.

[7] J. Vöge and M. Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *Proc. 12th Int. Conf. on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 202–215. Springer, 2000.
[8] S. Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *17th Annual Conference on Computer Science Logic (CSL 2008)*, 2008.
[9] Oliver Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS*, pages 145–156, 2009.