International Journal of Foundations of Computer Science
Vol. 21, No. 3 (2010) 277–287

DOI: 10.1142/S0129054110007246

World Scientific
www.worldscientific.com

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

OLIVER FRIEDMANN*

Institut für Informatik, Ludwig-Maximilians-Universität
Oettingenstraße 67, D-80538 München, Germany
Oliver.Friedmann@googlemail.com
http://www.tcs.ifi.lmu.de/∼friedman

Received 13 October 2008
Accepted 28 September 2009
Communicated by Jacques Sakarovitch

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 backend 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

*346 East 51st, 10022 NY

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[a] 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.

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 $w \in v_n E$ 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 $v_0 v_1 v_2 \dots$ 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 $\pi = v_0 v_1 v_2 \dots$ 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

[a] At least to our knowledge.

all sequences $v_0 \dots v_n$ with $v_{j+1} \in v_j E$ 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: $\sigma(v_0 \dots v_n) \in v_n E$. A play $v_0 v_1 \dots$ *conforms* to a strategy σ for player i if for all $j \in \mathbb{N}$ we have: if $v_j \in V_i$ and $v_0 v_1 \dots v_j \in \mathtt{dom}(\sigma)$ 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 $v_0 v_1 \dots$ with $v_0 = v$ conforming to σ and every j with $v_j \in V_i$ holds: $v_0 v_1 \dots v_j \in \mathtt{dom}(\sigma)$.

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 determinancy 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.

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 as follows

$$i_\pi : p \mapsto |\{j < |\pi| \mid \Omega(\pi(j)) = p \wedge \forall k > j . \Omega(\pi(k)) \leq p\}|$$

and it basically denotes for each priority p how often it occurs in π without seeing any greater priorities afterwards.

Let $\mathbf{0}$ denote the index that maps every priority to 0. The index of a play can be calculated iteratively by using 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 $i_\pi = (\ldots(\mathbf{0} \oplus \Omega(\pi(0))) \oplus \ldots) \oplus \Omega(\pi(|\pi| - 1))$.

Two indexes i and j (representing descriptions of two plays) can be compared with each other w.r.t. to their usefulness to a certain player $u \in \{0, 1\}$. Then $i >_u j$ holds iff there is some $k \in \mathtt{ran}(\Omega_G)$ s.t.

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

Sometimes, we are not only interested whether the relation $i >_u j$ holds, but also by which uniquely determined k. In this case, we write $i >_u^k j$.

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 $i_\pi >_1 i_{\pi'}$.*

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

(1) If $i <_p^w j$, $i <_p^q k$ and $\Omega(q) < \Omega(w)$ it follows that $k <_p^w j$.
(2) If $j <_p^w i$, $k <_p^q i$ and $\Omega(q) < \Omega(w)$ it follows that $j <_p^w k$.
(3) If $i <_p^w j$, $j <_p^q k$ and $u \in \{w, q\}$ s.t. $\Omega(w) \leq \Omega(u)$ and $\Omega(q) \leq \Omega(u)$ it follows that $i <_p^u k$.

As a general notation, $f[x \mapsto y]$ for a given function f, denotes the *f-update at x to y* which is formally defined as follows:

$$f[x \mapsto y](z) := \begin{cases} f(z) & \text{if } z \in \mathtt{dom}(f) \setminus \{x\} \\ y & \text{if } z = x \end{cases}$$

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 timestamping.

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 construction does not rely on the mechanism of dependency since it never happens that decisions have to be invalidated.

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 $k < |l|$ it holds that $l_k = (v, i, t, b, \_)$ implies $l'_k = (v, i, t, b, \_)$[b].

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 timestamp 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 timestamp. 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.

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,

[b] An occurrence of _ in a tuple is to be seen as an unbound existenially quantified variable.

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.

A complete pseudo-code outline of the algorithm is given in Fig. 1.

$\mathrm{decide}_G(v) =$
  `return` $\mathrm{explore}_G(v,\ \mathbf{0} \oplus \Omega(v),\ [],\ 1,\ (\mathbf{e}, \mathbf{e}))$

$\mathrm{explore}_G(v,\ i,\ l,\ c,\ d) =$
  `if` $(\exists p \in \{0,1\}.\ \exists (j, \_, \_) \in d_p(v) : i \geq_p j)$ `then`
    `return` $\mathrm{backtrack}_G(v,\ p,\ l,\ c+1,\ d)$
  `else`
    `if` $(\exists i < |l| : l_i = (v, j, t, b, a))$ `then`
      `return` $\mathrm{backtrack}_G(v,\ p,\ l[i \mapsto (v, j, t, b, a \cup \{p\})],\ c+1,\ d)$
      `where` $i >_p j$ [c]
    `else`
      `return` $\mathrm{explore}_G(w,\ i \oplus \Omega(w),\ (v, i, vE \setminus \{w\}, c, \emptyset) :: l,\ c+1,\ d)$
      `where` $w = \mathrm{select}_G(v,\ vE)$

$\mathrm{backtrack}_G(v,\ p,\ l,\ c,\ (d_0, d_1)) =$
  `if` $(l = [])$ `then`
    `return` $(p,\ \sigma_{d_p})$
  `else`
    `let` $l = (w, i, t, b, a) :: m$ `in`
    `if` $(w \in V_p)$ `or` $(t = \emptyset)$ `then`
      `return` $\mathrm{backtrack}_G(w,\ p,\ m,\ c,\ (d'_0, d'_1))$
      `where` $d'_p = d_p[w \mapsto d_p(w) \cup \{(i, c, u)\}]$

$$u = \begin{cases} \bot & \text{if } t = \emptyset \\ v & \text{otherwise} \end{cases}$$

$$d'_{1-p} = \begin{cases} y \mapsto \{(j, s, z) \in d_{1-p}(y) \mid j < b\} & \text{if } (1-p) \in a \\ d_{1-p} & \text{otherwise} \end{cases}$$

    `else`
      `return` $\mathrm{explore}_G(u,\ i \oplus \Omega(u),\ (w, i, t \setminus \{u\}, b, a) :: m,\ c,\ (d_0, d_1))$
      `where` $u = \mathrm{select}_G(w,\ t)$

Fig. 1. Modelchecking algorithm.

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

Due to the fact that every known efficient heuristic that tries to select a promising successor node can be easily fooled, we follow the most natural selection policy that randomly and uniformly selects a successor node:

$$\text{select}_G^{\mathcal{R}}(v,R) = u \in R \text{ with probability } \frac{1}{|R|}$$

Employing this selection policy, we will prove that the expected number of steps is at least exponential on a certain game. Nevertheless, for other reasonable policies, e.g. that always select an available successor with the highest priority, it should be also possible to show that a lower bound on the expected number of steps is also exponential.

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

4. Exponential Lower Bound

The disadvantage of the algorithm clearly lies in the index-structure that is used to compare the applicability of decisions. Basically, a decision at a point v with index i for player p tells us that if a play arrives at v with an associated index j that is at least as good as i, it is safe to assume that player p wins the play.

Intuitively, the index is a window to the past of a play - but in order to determine whether a decision can be reused to win another play arriving at the decision, it would be more precise to consider an index of what is lying ahead when following the decision-associated strategy. We exploit this observation in the following.

We present a family of parity games on which the expected runtime of the Model Checking Algorithm is exponential. The games are denoted by G_n. The set of nodes are $\mathcal{V}_n := \{a_0,\ldots,a_n,b_1,\ldots,b_n,c_1,\ldots,c_n\}$; all nodes are owned by player 1.

The priorities and edges have been outlined in Table 1; G_2 is depicted in Fig. 2.

Table 1. Edges and priorities.

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	$2 \cdot i - 1$	$\{a_{i-1}\}$

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

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

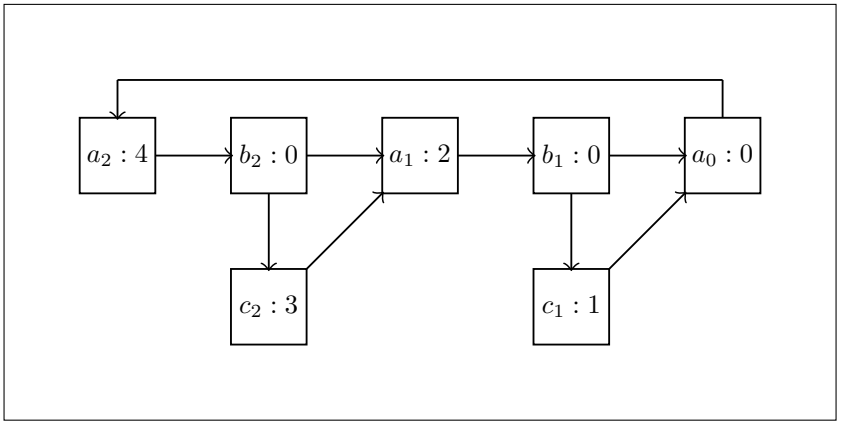

Fig. 2. The Game G_2.

Obviously, G_n is always 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 behaviour on these games is enforced as follows: assume 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 behaviour 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 are 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 $q \in \mathcal{V}_n$

- $\Phi_n(j, i, d)$ 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 $q \in \mathcal{V}$ and all G_n-indexes 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 $q \in \mathcal{V}$ and all G_n-indexes k
  (b) $(i, \_, \_) \in d_0(a_j)$

As a measure of the number of steps that the algorithm requires to solve the games, we simply use the time-counter that is maintained by the explore$_G$-routine. The following function f_n will be shown to capture the progression of it accurately. Let $n \in \mathbb{N}$ and $i < n$.

$$f_n(0) = 1 \qquad \text{and} \qquad f_n(i+1) = e_n(i) \cdot f_n(i) + 4$$

where

$$e_n(i) = \begin{cases} 1 & \text{if } \text{select}_{G_n}(b_i, b_i E) = c_i \\ 2 & \text{otherwise} \end{cases}$$

Lemma 5. *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$_{G_n}(a_j, i, l, c, d)$ *eventually 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 and the use of Corollary 2. □

Corollary 6. *Deciding a_n in G_n using* decide$_{G_n}$ *requires $f_n(n)+1$* explore$_{G_n}$*-steps.*

Proof. Calling decide$_{G_n}(a_n)$ directly invokes 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 5 implies that eventually backtrack$_{G_n}(a_n, 0, [], f_n(n) + 1, d')$ with some decisions d' is called. By definition of backtrack, it follows that the algorithm directly terminates, hence it totally requires $f_n(n) + 1$ explore-steps. □

Theorem 7. *Deciding a_n on G_n using* decide$_{G_n}$ *with* select$^{\mathcal{R}}_{G_n}$ *requires an expected number of $9 \cdot 1.5^n - 7$* 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 Corollary 6, it requires $f_n(n) + 1$ explore-steps to decide a_n. Since select$^{\mathcal{R}}_{G_n}$ uniformly selects edges, the expected number of steps $\bar{f}_n$ can be written

as follows:

$$\begin{aligned} \bar{f}_n(0) &= 1 \\ \bar{f}_n(i+1) &= \frac{1}{2} \cdot (1 \cdot \bar{f}_n(i) + 4) + \frac{1}{2} \cdot (2 \cdot \bar{f}_n(i) + 4) \\ &= \frac{3}{2} \cdot \bar{f}_n(i) + 4 \end{aligned}$$

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] O. Friedmann and M. Lange. Solving parity games in practice. Submitted, 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] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. To appear in LICS 2009, 2009.