A Brief Excursion to Parity Games

Bakhadyr Khoussainov(✉)

Department of Computer Science, University of Auckland, Auckland, New Zealand
bmk@cs.auckland.ac.nz

Abstract. The author in collaboration with Calude, Jain, Li and Stephan provided an algorithm that solves the parity games problem in quasi-polynomial time. In terms of running time, this is currently the most efficient algorithm that solves the parity games problem. The goal of this lecture is to give a brief background to the problem and present the algorithm in a somewhat informal way with a bit more emphasis on ideas rather than formal details.

1 Introduction

The emergence of parity games has been a complex and fascinating process. The roots of parity games go back to the works of Buchi [6,7], Rabin [26], Gurevich and Harrington [15] and McNaughton [23]. Numerous number of papers have been written on the subject, many results have been presented in conferences, and research networks and workshops have been organised. The general interest in parity games comes due to the following reasons. All these reasons are intimately inter-connected:

1. **Complexity-theoretic status**: Parity games problem is known to be in complexity classes NP and $coNP$. The problem is among a few algorithmic problems that belong to NP and $coNP$ but not known to be in P.
2. **Connections to modal μ-calculus**: Kozen invented modal μ-calculus [20]. This is an extension of propositional modal logic. The model checking problem consists of designing an algorithm that, given a formula ϕ and a transition labeled graph G, decides if G satisfies ϕ. It turns out that the model checking problem is polynomial time equivalent to solving the parity games problem [12].
3. **Connections to computer-aided verification**: The modal μ-calculus can be used as a formal specification language for expressing properties of reactive programs. Hence, many model checking problems in computer aided verification can be effectively reduced to parity games problem [4,5,12].
4. **Connections to automata and logic**: There are deep historical and mathematical connections between automata, two player games, modal μ-calculus, and parity games [28–30]. A recent example is in [9] that implements Kupferman-Vardi algorithm from [21]. The algorithm solves the parity games problem via a reduction to the emptiness problem of alternating parity automata.

M. Hoshi and S. Seki (Eds.): DLT 2018, LNCS 11088, pp. 24–35, 2018.
https://doi.org/10.1007/978-3-319-98654-8_3

To define parity games we need a finite directed bipartite graph $G = (V; E)$ which is given by its bipartite partition $V = V_1 \cup V_1$ and the set of edges $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$. We postulate one condition on the set E of edges that each v from V has at least one outgoing edge from v. Elements $v \in V$ are called *positions* of the game.

Definition 1. *An arena is given by the pair (G, p), where G is a bipartite graph as above and $p : V \to \omega \setminus \{0\}$ is a function called the priority function. For each $v \in V$, the value $p(v)$ is called the priority of v.*

Given an arena (G, p), there are two players: Player 0 and Player 1. Positions in V_0 are called Player 0 positions, and positions in V_1 are called Player 1 positions. We are also given the starting position $s \in V$, say $s \in V_0$. Player 0 and Player 1 play the parity game as follows. Imagine a token is placed on s. Player 0 moves the token along an edge outgoing from $v_0 = s$ and moves to position v_1. Player 1 now moves the token along an edge outgoing from v_1 and moves to position v_2. Player 0 moves the token along an edge outgoing from v_2 and moves to position v_3. This continues on. If the start position s is in V_1 then Player 1 makes the first move. Thus, the players produce an infinite path in G called a *play*:

$$\rho = v_1, v_2, v_3, v_4, \ldots, v_i, v_{i+1}, \ldots$$

Definition 2. *Player 0 wins the play ρ if the maximal priority of positions that appear infinitely often in ρ is even. Otherwise, Player 1 wins the play.*

Clearly, every play ρ of the parity game played on (G, p) is won by either Player 0 or Player 1. An important fact, however, is this. One of the players has always a winning strategy. In other words, one of the players has a strategy f_s (the index indicates the starting position) such that all plays consistent with f_s are won by the player. This fact is known as a determinacy of parity games.

Determinacy of parity games is not surprising. For instance, Martin's determinacy theorem states that all games with Borel winning conditions are determined [22]. The winning condition for parity games is a Borel condition. Therefore, we can partition that set V of positions of the arena (G, p) into sets W_0 and W_1:

$$W_0 = \{v \in V \mid \text{Player 0 wins the game starting at } v\},$$
$$W_1 = \{v \in V \mid \text{Player 1 wins the game starting at } v\}.$$

Importantly, parity games are effectively determined. There exists an algorithm that given a parity game (G, p) and starting position s, tells us which of the players is the winner of the game. Thus, we have the following parity games problem. The statement of the problem is due to effective determinacy of parity games.

Design an efficient algorithm that, given a parity game played on (G, p), outputs the sets W_0 and W_1 of winning positions for Player 0 and Player 1.

There are numerous algorithms that solve parity games. Emerson and Jutla [12] showed that the problem is in $NP \cap co\text{-}NP$, and Jurdzinski [17] improved this bound to $UP \cap co\text{-}UP$. Jurdzinski also provided small progress measure algorithm for solving parity games. Petersson and Vorobyov [25] devised a subexponential randomised algorithm. Later, Jurdzinski, Paterson and Zwick [19] also designed a deterministic algorithm of similar complexity. McNaughton [23] showed that the winner of a parity game can be determined in time $n^{c+O(1)}$, where c is the maximal priority of the positions. There is also Zielonka's recursive algorithm [31] which is considered to be useful in practice. Another interesting algorithm is by Schewe [27]. There are also various subclasses of parity games or their relatives which can be solved in polynomial time [2,3,14,16,24]. The author in collaboration with Calude, Jain, Li and Stephan provided an algorithm that runs in time $O(n^{\log(c)+3})$. In terms of running time, this is currently the most efficient algorithm that solves the parity games problem. The goal is to present the algorithm in somewhat informal way with an emphasis on ideas rather than formal details.

2 Memoryless Winning Strategies

A crucial property of parity games is that the winner has a *memoryless winning strategy*. This is known as *memoryless determinacy* of parity games. A memoryless strategy is defined by the current position of the play, and does not depend on the history of the play at all. Formally, a memoryless strategy for Player ϵ is given by a function $f : V_\epsilon \to V_{\epsilon+1}$ such that $(v, f(v)) \in E$ for all $v \in V_\epsilon$. There are many proofs of memoryless determinacy, see for instance [1,12,23,31]. Below we provide a simple proof of this result, the simplest possible proof known to the author. The proof is non-constructive but implies memoryless determinacy.

Theorem 3 (Memoryless Determinacy). *The winner in any parity game (G, p) has a memoryless winning strategy.*

Proof. Let $s \in V_1$ be a starting position such that the set $X = \{u \mid (s, u) \in E\}$ is not a singleton. Let X_1, X_2 be a non-trivial partition of X. We *split* (G, p) into two parity games (G_1, p) and (G_2, p), where G_1 is the same as G but Player 1 at s can only move to X_1, and G_2 is the same as G but Player 1 at position s can only move to X_2.

We claim that Player 0 wins G starting at s iff Player 0 wins both G_1 and G_2 starting at s. Indeed, if Player 0 wins G, then clearly Player 0 wins both G_1 and G_2. For the other direction, assume that Player 0 wins both G_1 and G_2 with winning strategies f_1 and f_2, respectively. Below we describe a winning strategy f for Player 0 in (G, p).

Let $\gamma = v_0, v_1, \ldots, v_i$ be a finite play in G, and $v_i, v_{i+1}, \ldots, v_j$ be a segment of the play such that $i < j$, $v_i = s$ and no position among $v_{i+1}, \ldots, v_j$ equals s. Call such segments *proper segments*. If $v_{i+1} \in X_1$, then say that the segment is in G_1; otherwise we say that the segment is in G_2. Consider α obtained from γ by removing all proper segments that occur in G_2. Likewise, let β be obtained

from γ by removing all proper segments that occur in G_1. The strategy f is now defined as follows. The strategy f follows f_1 by setting $f(\gamma) = f_1(\alpha)$ if the last proper segment occurs in G_1, and the strategy f follows f_2 by setting $f(\gamma) = f_2(\beta)$ if the last proper segment occurs in G_2. It is clear that f is a winning strategy for Player 0 in game (G, p).

Apply now the splitting process above to the arenas (G_1, p) and (G_2, p) until we reach games (G', p) such that the set of positions of G' coincide with the set of positions of G, and the set E' of edges is such that $E' \subseteq E$ and Player 1 has exactly one outgoing edge from any position v in V_1. Let us call such games *solitary games* for Player 1.

We conclude that Player 1 wins the parity game starting at s iff Player 1 wins a solitary game starting at s. Each solitary game for Player 1 induces a memoryless strategy.

One applies the arguments above, including the notion of splitting, for Player 0. That implies that Player 0 wins the parity game from the start position s if and only if Player 0 wins one of Player 0's solitary games starting at position s. □

There is a polynomial time algorithm that, given a memoryless strategy for Player ϵ and position s, detects if the strategy is a winning strategy from s. Hence, we have:

Corollary 4. *Parity games problem is in NP and coNP.* □

3 Reachability Games

Reachability games are well-studied. We mention these games since we reduce the parity games problem to the reachability games problem.

Just like parity games, a reachability game is given by Player 0 and Player 1, a bipartite directed graph $G = (V, E)$, a starting position $s \in V$, and Player 0's *target set* $T \subseteq V$ of positions. Starting at s, the players move alternatively and produce a path $\rho = v_0, v_1, \ldots, v_i, \ldots$, where $v_0 = s$. Note that if $s \in V_0$ then Player 0 starts the play, and otherwise Player 1 starts the play. We say that Player 0 *wins the play* ρ if ρ meets the target set T, that is, there is a position v_i in the play such that $v_i \in T$.

Theorem 5 (Folklore: Determinacy of Reachability Games). *There is a linear time algorithm that given a reachability game finds the winner of the game.*

Proof. Consider a reachability game as described above. The idea is to start computing vertices, inductively, from which Player 0 can force plays into the target set T. Clearly, no move is required for Player 0 to win the game if the game starts at positions from T. So, we set $T_0 = T$. Assume that we defined the set T_i. Intuitively, T_i consists of all v such that starting at v Player 0 can force plays into T in at most i moves. By induction, we also have $T_0 \subset T_1 \subset \ldots \subset T_i$.

To build T_{i+1}, put all v in T_i into T_{i+1} and for each $v \notin T_i$ proceed as follows. If $v \in V_0$ and there is an edge $(v, u) \in E$ with $u \in T_i$ then we put v into T_{i+1}. Also, if $v \in V_1$ and $u \in T_i$ for all $(v, u) \in E$ then we put v into T_{i+1}. This defines T_{i+1}. Clearly Player 0 wins the reachability game starting at any position $v \in T_{i+1}$.

Thus, we have the sequence $T_0 \subset T_1 \subset T_2 \subset \ldots$. Let X be the union of this chain. Player 0 wins the game from s if and only if $s \in X$. One can implement the algorithm above that runs in linear time. □

4 A Quasipolynomial Time Algorithm for Parity Games

For this section, let us fix (G, p) the arena of a parity game with n positions. Let c be the maximal priority, that is, the maximal value in the range of the priority function p. Recall that $p(v) > 0$ for all $v \in V$. We can also always assume that $p(v) \leq n$ for all $v \in V$. For this section we postulate that all positions in V_0 have even priorities (that is, $p(v)$ is even for all $v \in V_0$), and all positions in V_1 have odd priorities (that is, $p(v)$ is odd for all $v \in V_1$). This is an acceptable postulate since every parity game can be reduced in linear time to parity games where priorities of Player 0 positions are even, and priorities of Player 1 positions are odd. The aim in this section is to reduce parity games problem to solving reachability games.

4.1 A Naive Reduction

The winning condition in the parity game for a given play ρ is an infinitary condition. Namely, one needs to consider all positions in the play ρ that appear infinitely often and then consider the parity of the maximal priority among these positions. If the parity is even, then Player 0 wins the play; otherwise Player 1 wins the play. This winning condition can be turned into a finitary definition of winning in a finite game as follows. The players, Player 0 and Player 1, play just like in the parity game on (G, p) and start producing a sequence $v_0, v_1, \ldots, v_j$ of moves. The players stop as soon as the last position v_j in the play repeats itself, that is, $v_j = v_i$ for some $i < j$ in the play. Then declare Player 0 to be the winner if the maximal priority that appears in the cycle $v_i, v_{i+1}, \ldots, v_{j-1}, v_j$ is even. It is easy to see that the game thus defined is a reachability game. A bit more work is needed to prove that this reachability game is equivalent to the original parity game in the arena (G, p). The number of positions in the reachability game is bounded by $n!$ from below. Hence, this naive reduction of parity games to reachability games is clearly inefficient.

4.2 Unsuccessful Attempt: Good Positions

This subsection is somewhat intuitive and informal. All the notions here can be made formal. The aim is to present some intuition needed for the concepts to be defined in the next subsection.

Let $\rho = v_0, v_1, \ldots$ be a play in parity game. Assume that Player 0 wins the play. Then we can write the play ρ as $\alpha\beta_1\beta_2\beta_3\ldots$ such that in each segment β_i the maximal priority is even and it appears in the last position of β_i. We now attempt to formalise this simple observation. A finite play $\alpha = v_1, \ldots, v_k$, $k > 1$, is *good* if the maximum priority position in this play is even and it is achieved at v_k. A position v is *good* if Player 0 can guarantee a good play starting at v. It is not too hard to see that one computes in linear time all good positions for Player 0. For instance, this can be done using ideas similar to the algorithm that solves the reachability game problem.

The concept of good position can now be iterated. For a given integer $i > 0$, we say that a play α is *i-good* if α can be written as $\beta_1\beta_2\ldots\beta_i$ so that each segement β_i is a good play. A position v is *i-good* if Player 0 can guarantee an i-good play starting at v. Note that any $(i+1)$-good position is also i-good. Now we can collect all positions that are i-good for all i. Further, a position is ω-good if it is i-good for all $i \geq 1$. Note that one can similarly define the notion of i-good and ω-good positions for Player 1.

The following is not too hard to see. Player 0 wins the parity game from starting at any of the ω-good positions. These ω-good positions can be computed in polynomial time. Interestingly, we still have a mess. Namely, there are non-empty parity games such that no player in the games possesses ω-good positions.

4.3 Ghost Sequences and Statistics

We want to code good plays in more sophisticated way. Let us fix a play

$$\rho = v_0, v_1, v_2, \ldots$$

in the parity game (G, p). Recall that c denotes the maximal value in the range of p. Also, recall that $p(v) > 0$ for all $v \in V$. A *segment* of play ρ is any consecutive sequence of moves in ρ. So, each segment is of the form $v, v_{i+1}, \ldots, v_j$, where $i \leq j$.

Definition 6. *A ghost i-sequence is a sub-sequence $a_1, a_2, a_3, \ldots, a_s$ of the play ρ such that that (1) $s = 2^i$, and (2) in the play between a_k and a_{k+1} the maximum priority is even, where $1 \leq k < s$.*

The subsequence $a_1, a_2, a_3, \ldots, a_s$ in the definition does not have to be a sequence of consecutive moves in the play ρ. Later we clarify why we name the subsequences $a_1, a_2, a_3, \ldots, a_s$ ghost subsequences. Note that the definition is given for Player 0. Ghost sequences for Player 1 are defined similarly by simply changing the parity.

The goal is to design a mechanism that detects ghost $(log(n)+1)$-sequences. The reason is that any such sequence has repeated positions, and the maximal priority in the play between these positions is even by the definition of the ghost sequence. Detecting ghost $(log(n)+1)$-sequences is done through the concept of statistics:

Definition 7 (Statistics). *A statistics of a player is a sequence* $(b_0, b_1, \ldots, b_{\log(n)+1})$ *such that* $0 \leq b_i \leq c$ *for all* i*, and if* b_i, b_j *are both non-zero and* $i < j$ *then* $b_i \leq b_j$*. A winning statistics is one with* $b_{\log(n)+1} \neq 0$*.*

So, non-zero members of the statistics form an increasing, not necessarily strictly increasing, sequence. Note that the space needed for each statistics is $O(\log(n)\log(c))$.

What does the statistics $(b_0, \ldots, b_i, \ldots, b_{\log(n)+1})$ *tell us?*

The answer is that each statistics are designed to detect existence of ghost sequences in a play. Below we describe the semantics of the statistics $(b_0, \ldots, b_i, \ldots, b_{\log(n)+1})$ that we call **invariants** of the statistics:

1. The initial statistics is $(0, 0, \ldots, 0)$. The initial statistics is $(0, 0, \ldots, 0)$ is the sate at which the players start to play.
2. Each b_i is a priority of a position that occurred in the play or $b_i = 0$.
3. Each b_i is associated with at most one ghost i-sequence. We denote the ghost i-sequence associated with b_i by $ghost(b_i)$.
4. The priority b_i is the *only* information about the ghost i-sequence $ghost(b_i)$. When $b_i > 0$, the priority b_i tells us that a ghost sequence $ghost(b_i)$ appeared in the play.
5. If $b_i > 0$, then the value b_i is the largest priority occurred at the end or after the ghost i-sequence $ghost(b_i)$.
6. If $b_i = 0$, then no ghost i-sequence is being associated with b_i. In other words, the sequence $ghost(b_i)$ is the nil sequence.
7. If b_i, b_j are both non-zero and $i < j$ then $b_i \leq b_j$.
8. If $0 < b_i \leq b_j$ and $i < j$, then the ghost i-sequence associated with b_i starts after a position with priority b_j was visited at or after the end of the ghost j-sequence.

When players are playing, statistics are being updated. So, consider the play

$$\alpha = v_0, v_1, \ldots, v_k$$

and let $(b_0, \ldots, b_i, \ldots, b_{\log(n)+1})$ be the current statistics of α. Each b_i is associated with the sequence $ghost(b_i)$ which is either the nil sequence or ghost i-sequence. The ghost sequence $ghost(b_i)$ is a subsequence of a segment of the play α. Denote the segment by $segment(b_i)$. When $ghost(b_i)$ is the nil-sequence, so is $segment(b_i)$. In terms of notations and explanations above, the idea is that the play α can be written as

$$\beta \cdot \ segment(b_{\log(n)+1})) \cdot \ldots \cdot segment(b_i) \cdot \ldots \cdot segment(b_1) \cdot segment(b_0),$$

where β is a prefix of α. The next move by a player triggers an update of the statistics so that the properties (1)–(8) stay true. The update is explained in the next subsection.

It is important to point out that given b_i we *cannot* recover the sequence $ghost(b_i)$ from b_i. We only know that the ghost sequence $ghost(b_i)$ exists, and

b_i is the only information about the ghost i-sequence. Moreover, updates might change the ghost sequence linked with b_i. These are the main reasons for the term *ghost*. Yet, we use the existence of ghost sequences in the arguments for correctness of our algorithm.

4.4 Updating Statistics

Consider the play $\alpha = v_0, v_1, \ldots, v_k$ and let $\bar{b} = (b_0, \ldots, b_i, \ldots, b_{\log(n)+1})$ be the current statistics of α. Let the next move, made by one of the players, be to position v_{k+1}. We describe two updates depending on the parity of the priority $p(v_{k+1})$ and present the reasoning for the updates.

Case 1. Assume that $p(v_{k+1})$ is even (and hence $v_{k+1} \in V_0$). Set $b = p(v_{k+1})$. Update the current statistics $\bar{b}$ as follows:

- Select the largest j such that either
  1. ($b_j = 0 \ \vee \ b_j$ is odd) and all b_i, where $i < j$, are non-zero and even,
  2. or $0 < b_j < b$

  and then set $b_j = b$ and $b_i = 0$ for all $i < j$.
- If this update produces a non-zero $b_{\log(n+1)}$ then Player 0 is declared a winner.

Reasoning:
Suppose the first rule is applied for the update. We have ghost sequences

$$ghost(b_{j-1}),\ ghost(b_{j-2}),\ \ldots,\ ghost(b_0)$$

that are present in the respective segments of the play α:

$$segment(b_{j-1}),\ segment(b_{j-2}),\ \ldots,\ segment(b_0).$$

Note that the concatenation

$$ghost(b_{j-1}) \cdot ghost(b_{j-2}),\ \ldots, \cdot ghost(b_0)v_{k+1}$$

is a ghost j-sequence that now we denote by $ghost(b_j)$ and the segment $segment(b_j)$ that corresponds to $ghost(b_j)$ becomes:

$$segment(b_{j-1}) \cdot segment(b_{j-2}) \cdot \ldots \cdot segment(b_0)v_{k+1}.$$

Now note that with these changes all the old ghost sequences $ghost(b_j)$, $ghost(b_{j-1})$, ... , $ghost(b_0)$ and their corresponding segments become nil sequences. It is not too hard to see that all the invariants (1)–(8) listed above are preserved.

Now suppose that the second rule is applied for the update. In this case, let us replace the last position of $ghost(b_j)$ with v_{k+1}. The result is a new ghost j-sequence $ghost(b_j)$ but the segment that corresponds to it is now

$$segment(b_{j-1}) \cdot segment(b_{j-2}) \cdot \ldots \cdot segment(b_0)v_{k+1}.$$

With these changes all the old ghost sequences $ghost(b_{j-1}), \ldots, ghost(b_0)$ and their corresponding segments become nil sequences. Again, it is not too hard to see that all the invariants (1)–(8) listed above are preserved.

Case 2. Assume that $p(v_{k+1})$ is odd (and hence $v_{k+1} \in V_1$). Set $b = p(v_{k+1})$. Update the current statistics $\bar{b}$ as follows. Select the largest j such that $0 < b_j < b$, and then set $b_j = b$ and $b_i = 0$ for all $i < j$.

Reasoning:
In this case the j-ghost sequence does not change but $segment(b_j)$ that corresponds to $ghost(b_j)$ becomes

$$segment(b_{j-1}) \cdot segment(b_{j-2}) \cdot \ldots \cdot segment(b_0)v_{k+1}.$$

With these changes all the old ghost sequences $ghost(b_{j-1}), \ldots, ghost(b_0)$ and their corresponding segments become nil sequences. Again, it is not too hard to see that all the invariants (1)–(8) listed above are preserved.

4.5 Reduction to Reachability Games

The notion of statistics and their updates (that we defined for the parity game on (G, p)) naturally lead to consider the reachability G_R game based on (G, p) with a starting position s. Here is a description of the reachability game G_R:

1. The positions of G_R are $(a, \tilde{b})$, where $a \in G$ and $\tilde{b}$ is a statistics.
2. The starting position is $(s, \tilde{0})$, where s is the starting position of the parity game.
3. The target set T is the set of all pairs $(a, \tilde{b})$ such that $\tilde{b}$ indicates a win.
4. Player 0 can move from $(a, \tilde{b})$ to $(a', \tilde{b}')$ if and only if $a \in V_0$ and the move from a to a' in G causes the statistics $\tilde{b}$ to be updated to $\tilde{b}'$, and $\tilde{b}$ does not indicate a win.
5. Player 1 can move from $(a, \tilde{b})$ to $(a', \tilde{b}')$ if and only if $a \in V_1$ and the move from a to a' in G causes the statistics $\tilde{b}$ to be updated to $\tilde{b}'$, and $\tilde{b}$ does not indicate a win.

Note that the reachability game G_R is defined with respect to Player 0. One can defined the corresponding reachability game for Player 1.

Theorem 8 ([8]). *Player 0 wins parity games played on (G, p) starting at s if and only if Player 0 wins the reachability game G_R.*

Proof (Outline). The proof follows from the following sequence of claims. Our claims here are for Player 0. Similar arguments can be done for Player 1.

Claim 1: *If Player 0 wins the game G_R then the player wins the parity game G.*

To prove the claim, consider a play in the parity game G that is consistent with the winning strategy for Player 0 in G_R. The play ends in the target set. So, the play is won by a ghost sequence $\tilde{b}$ being detected such that $b_{\log(n)+1} > 0$. Thus, for the play there is a ghost sequence $a_1, \ldots, a_s$ of length $2^{\log(n)+1}$ in which

$a_k = a_\ell$ for some $k \neq \ell$. The maximum priority b of a position between a_h and a_ℓ in the play must be even by the definition of a ghost i-sequence. Thus a loop has been observed for which the maximum priority of a position in the loop is even.

Claim 2: *Assume that Player 1 wins the parity game G. Then Player 0 can not win the reachability game G_R.*

To prove the claim, suppose that Player 1 follows a memoryless winning strategy in the parity game G. Let us assume that Player 0 wins the reachability game G_R. Then Player 0 goes into a loop in G such that the maximum priority position in the loop is even. Since, Player 1 is using memoryless winning strategy in parity game, Player 0 can force Player 1 to stay in the loop forever and win the play in G. This is a contradiction since Player 1 used winning strategy.

Finally, we need to explain (but not prove) the next claim:

Claim 3: *Assume Player 0 wins the parity game G. If the player follows a memoryless winning strategy in the parity game then the player wins the reachability game G_R.*

The proof is somewhat routine but not hard. One way to check the proof is to imagine the following situation. Assume that Player 0 follows a memoryless winning strategy in G. Any play consist with the strategy then arrives to a cycle such that the maximal priority position in this cycle is even. Note that Player 1 does not have to stay in this cycle. But if Player 1 decides to stay in the cycle forever then one needs to analyse the way statistics changes along the play (due to the maximal priority position appearing in the play). So, one can show, in this particular case, that eventually Player 0 reaches the target set in G_R. □

Now we would like to compute the size of the positions in the reachability game G_R. The size of the statistics (for Player 0) is given by $\log(n) + 2$ numbers each of size $\log(c)$. Note that $log(c) \leq \log(n)$. Therefore, overall size of a representation of a position in the reachability game G_R is bounded by $(\log(n)+2)(\log(c)+1)$. Hence, the number of positions in the reachability game G_R is in order $O(n^{\log(c)+1})$. Since each position in the reachability game G_R has at most n outgoing edges, the number of edges in G_R is $O(n^{\log(c)+2})$. We already explained that the reachability games can be solved in liner time on the size of the of the games. Thus finding out if position s in the parity game (G, p) is a winning position for Player 0 takes time proportional to $O(n^{\log(c)+2})$. Hence, we have the following theorem that tells us the running time bound for the algorithm that outputs the winning sets W_0 and W_1 for the players:

Theorem 9 (Quasi-Polynomial Solution). *There is an algorithm that, given a parity game (G, p) with n positions, solves the game in time $O(n^{\log(c)+3})$.* □

The reduction of parity games to reachability games also provides fixed parameter tractability (FPT) result for solving parity games. For fixed parameter tractability see the textbook [10]. Here the parameter is c the maximal number of priorities. To get the FPT result we count the number of positions in G_R in a bit different way.

Consider the statistics $\tilde{b} = (b_0, b_1, \ldots, b_m)$, where $m = \log(n) + 1$. We would like to put a bound on the number of statistics so that the parameter n is removed from the counting. Say, for simplicity that each $b_i \neq 0$. Then by the definition of statistics we have $b_0 \leq b_1 \leq \ldots \leq b_m$. Call such statistics strict. We can transform the strict statistics into the following sequence:

$$(b_0, b_1 + 1, b_2 + 2, \ldots, b_i + i, \ldots, b_m + m).$$

The mapping $(b_0, b_1, \ldots, b_m) \rightarrow (b_0, b_1+1, b_2+2, \ldots, b_i+i, \ldots, b_m+m)$ induces an injection from the set of all strict statistics into the power set $\{0, \ldots, 2c\}$. Hence, there are at most 2^{2c} strict statistics. We removed the dependence on n. One can code up all statistics in a similar way, and construct an injection map from the set of all statistics into the power set $\{0, \ldots, 3c\}$. Therefore one can prove that the number of all statistics is bounded by 2^{3c}. This implies that the number of all positions in the reachability game G_R is bounded by $n \cdot 2^{2c}$. Since each position in the reachability game G_R has at most n outgoing edges, the number of edges in G_R is bounded by $2^{2c} \cdot n^2$. Thus, from determinacy of reachability game, we have the following theorem:

Theorem 10 (FPT Theorem). *There is an algorithm that, given a parity game with n positions, solves the game in time proportional to $g(c) \cdot n^3$, where $g(c) = 2^{2c}$.* □

References

1. Björklund, H., Sandberg, S., Vorobyov, S.: Memoryless determinacy of parity and mean payoff games: a simple proof. Theor. Comput. Sci. **310**(1–3), 365–378 (2004)
2. Bodlaender, H.L., Dinneen, M.J., Khoussainov, B.: On game-theoretic models of networks. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 550–561. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45678-3_47
3. Bodlaender, H.L., Dinneen, M.J., Khoussainov, B.: Relaxed update and partition network games. Fundamenta Informaticae **49**(4), 301–312 (2002)
4. Bernet, J., Janin, D., Walukiewich, I.: Permissive strategies from parity games to safety games. ITA **36**(3), 261–275 (2002)
5. Bradfield, J., Walukiewicz, I.: The mu-calculus and model checking. Handbook of Model Checking, pp. 871–919. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_26
6. Richard Büchi, J.: On a decision method in the restricted second order arithmetic. In: Proceedings of the International Congress on Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1960)
7. Richard Büchi, J.: Decision methods in the theory of ordinals. Bull. Am. Math. Soc. **71**, 767–770 (1965)
8. Calude, C.S., Jain, S., Khoussainov, B., Li, W., Stephan, F.: Deciding parity games in quasipolynomial time. Proc. STOC **2017**, 252–263 (2017)
9. Di Stasio, A., Murano, A., Perelli, G., Vardi, M.Y.: Solving parity games using an automata-based algorithm. In: Han, Y.-S., Salomaa, K. (eds.) CIAA 2016. LNCS, vol. 9705, pp. 64–76. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40946-7_6

10. Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity Theory. Springer, Heidelberg (2013). https://doi.org/10.1007/978-1-4471-5559-1
11. Friedmann, O.: An exponential lower bound for the parity game strategy improvement algorithm as we know it. In: Logic in Computer Science, LICS vol. 2009, pp. 145–156 (2009)
12. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Annals of IEEE Symposium on Foundations of Computer Science, pp. 368–377 (1991)
13. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model checking for the μ-calculus and its fragments. Theor. Comput. Sci. **258**(1–2), 491–522 (2001)
14. Gandhi, A., Khoussainov, B., Liu, J.: Efficient algorithms for games played on trees with back-edges. Fundamenta Informaticae **111**(4), 391–412 (2011)
15. Gurevich, Y., Harrington, L.: Trees, automata and games. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC 1982, 5–7 May 1982, San Francisco, California, USA, pp. 60–65 (1982)
16. Ishihara, H., Khoussainov, B.: Complexity of some infinite games played on finite graphs. In: Goos, G., Hartmanis, J., van Leeuwen, J., Kučera, L. (eds.) WG 2002. LNCS, vol. 2573, pp. 270–281. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36379-3_24
17. Jurdzinski, M.: Deciding the winner in parity games is in UP ∩ co -UP. Inf. Process. Lett. **68**(3), 119–124 (1998)
18. Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46541-3_24
19. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. SIAM J. Comput. **38**(4), 1519–1532 (2008)
20. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. **27**(3), 333–354 (1983)
21. Kupferman, O., Vardi, M.Y.: Weal alternating automata and tree automata emtiness. In: STOC, pp. 224–233 (1998)
22. Martin, D.A.: Borel determinacy. Ann. Math. Second Ser. **102**(2), 363–371 (1975)
23. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Logic **65**(2), 149–184 (1993)
24. Obdrzalek, J.: Algorithmic analysis of parity games. Ph.D. thesis, University of Edinburgh (2006)
25. Petersson, V., Vorobyov, S.G.: A randomized subexponential algorithm for parity games. Nordic J. Comput. **8**, 324–345 (2001)
26. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Am. Math. Soc. **141**, 1–35 (1969)
27. Schewe, S.: Solving parity games in big steps. J. Comput. Syst. Sci. **84**, 243–262 (2017)
28. Stirling, C.: Bisimulation, modal logic and model checking games. Logic J. IGPL **7**(1), 103–124 (1999)
29. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-59042-0_57
30. Wilkie, T.: Alternating tree automata, parity games and modal μ-calculus. Bull. Belg. Math. Soc. **8**(2), 359–391 (2001)
31. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. **200**, 135–183 (1998)