Solving Parity Games via Priority Promotion

Massimo Benerecetti[1], Daniele Dell'Erba[1], and Fabio Mogavero[2*]

[1] Università degli Studi di Napoli Federico II
[2] Oxford University

Abstract. We consider *parity games*, a special form of two-player infinite-duration games on numerically labelled graphs, whose winning condition requires that the maximal value of a label occurring infinitely often during a play be of some specific parity. The problem has a rather intriguing status from a complexity theoretic viewpoint, since it belongs to the class UPTime ∩ CoUPTime, and still open is the question whether it can be solved in polynomial time. Parity games also have great practical interest, as they arise in many fields of theoretical computer science, most notably logic, automata theory, and formal verification. In this paper, we propose a new algorithm for the solution of the problem, based on the idea of promoting vertices to higher priorities during the search for winning regions. The proposed approach has nice computational properties, exhibiting the best space complexity among the currently known solutions. Experimental results on both random games and benchmark families show that the technique is also very effective in practice.

1 Introduction

Parity games [45] are perfect-information two-player turn-based games of infinite duration, usually played on finite directed graphs. Their vertices, labelled by natural numbers called *priorities*, are assigned to one of two players, named *Even* and *Odd* or, simply, 0 and 1, respectively. The game starts at an arbitrary vertex and, during its evolution, each player can take a move only at its own positions, which consists in choosing one of the edges outgoing from the current vertex. The moves selected by the players induce an infinite sequence of vertices, called play. If the maximal priority of the vertices occurring infinitely often in the play is *even*, then the play is winning for player 0, otherwise, player 1 takes it all.

Parity games have been extensively studied in the attempt to find efficient solutions to the problem of determining the winner. From a complexity theoretic perspective, this decision problem lies in NPTime ∩ CoNPTime [18, 19], since it is *memoryless determined* [17, 37, 38, 45]. It has been even proved to belong to UPTime ∩ CoUPTime [31], a status shared with the *factorisation problem* [1, 24, 25]. They are the simplest class of games in a wider family with similar complexities and containing, *e.g.*, *mean payoff games* [16, 30], *discounted payoff games* [55], and *simple stochastic games* [15]. In fact, polynomial time reductions

* Sponsored by the Engineering and Physical Sciences Research Council of the United Kingdom, grant EP/M005852/1.

exist from parity games to the latter ones. However, despite being the most likely class among those games to admit a polynomial-time solution, the answer to the question whether such a solution exists still eludes the research community.

The effort devoted to provide efficient solutions stems primarily form the fact that many problems in formal verification and synthesis can be reformulated in terms of solving parity games. Emerson, Jutla, and Sistla [18, 19] have shown that computing winning strategies for these games is linear-time equivalent to solving the modal μCALCULUS model checking problem [20]. Parity games also play a crucial role in automata theory [17, 36, 44], where, for instance, they can be applied to solve the complementation problem for alternating automata [29] and the emptiness of the corresponding nondeterministic tree automata [36]. These automata, in turn, can be used to solve the satisfiability and model checking problems for expressive logics, such as the modal [53] and alternating [2, 51] μCALCULUS, ATL* [2, 50], Strategy Logic [14, 40, 41, 43], Substructure Temporal Logic [4, 5], and fixed-point extensions of guarded first-order logics [7, 8].

Previous solutions mainly divide into two families: those based on decomposing the game into subsets of winning regions, called dominions, and those trying to directly build winning strategies for the two players on the entire game. To the first family belongs the *divide et impera* solution originally proposed by McNaughton [39] for Muller games and adapted to parity games by Zielonka [54]. More recent improvements to that recursive algorithm have been proposed by Jurdziński, Paterson, and Zwick [33, 34] and by Schewe [48]. Both approaches rely on finding suitably closed dominions, which can then be removed from a game to reduce the size of the subgames to be recursively solved. To the second family belongs the procedure proposed by Jurdziński [32], which exploits the connection between the notions of progress measures [35] and winning strategies. An alternative approach was proposed by Jurdziński and Vöge [52], based on the idea of iteratively improving an initial non-winning strategy. This technique was later optimised by Schewe [49]. From a purely theoretical viewpoint, the best asymptotic behaviour obtained to date is the one exhibited by the solution proposed in [48], which runs in time $\mathrm{O}\left(e \cdot n^{\frac{1}{3}k}\right)$, where n and e are the number of vertices and edges of the underlying graph and k is the number of priorities. As far as space consumption is concerned, we have two incomparable best behaviours: $\mathrm{O}(k \cdot n \cdot \log n)$, for the small progress measure procedure of [32], and $\mathrm{O}\left(n^2\right)$, for the optimised strategy improvement method of [49]. Due to their inherent recursive nature, the algorithms of the first family require $\mathrm{O}(e \cdot n)$ memory, which could be reduced to $\mathrm{O}\left(n^2\right)$, by representing subgames implicitly through their sets of vertices. All these bounds do not seem to be amenable to further improvements, as they appear to be intrinsic to the corresponding solution techniques. Polynomial time solutions are only known for restricted versions of the problem, where one among the tree-width [22, 23, 46], the dag-width [6], the clique-width [47] and the entanglement [9] of the underlying graph is bounded.

The main contribution of the paper is a new algorithm for solving parity games, based on the notions of *quasi dominion* and *priority promotion*. A quasi dominion Q for player $\alpha \in \{0, 1\}$, called a *quasi α-dominion*, is a set of vertices from each

of which player α can enforce a winning play that never leaves the region, unless one of the following two conditions holds: *(i)* the opponent $\overline{\alpha}$ can escape from Q or *(ii)* the only choice for player α itself is to exit from Q (*i.e.*, no edge from a vertex of α remains in Q). Quasi dominions can be ordered by assigning to each of them a priority corresponding to an under-approximation of the best value the opponent can be forced to visit along any play exiting from it. A crucial property is that, under suitable and easy to check assumptions, a higher priority quasi α-dominion Q_1 and a lower priority one Q_2, can be merged into a single quasi α-dominion of the higher priority, thus improving the approximation for Q_2. For this reason we call this merging operation a priority promotion of Q_2 to Q_1. The underlying idea of our approach is to iteratively enlarge quasi α-dominions, by performing sequences of promotions, until an α-dominion is obtained.

We prove soundness and completeness of the algorithm. Moreover, a bound $\mathrm{O}\Big(e \cdot (3\frac{n-2}{k-2})^{k-1}\Big)$ on the time complexity and a $\mathrm{O}(n \cdot \log k)$ bound on the memory requirements are provided. Experimental results, comparing our algorithm with the state of the art solvers, also show that the proposed approach perform very well in practice, most often significantly better than existing ones, on both random games and benchmark families proposed in the literature.

2 Parity Games

Let us first briefly recall the notation and basic definitions concerning parity games that expert readers can simply skip. We refer to [3] [54] for a comprehensive presentation of the subject.

Given a partial function $\mathsf{f} : \mathrm{A} \rightharpoonup \mathrm{B}$, by $\mathsf{dom}(\mathsf{f}) \subseteq \mathrm{A}$ and $\mathsf{rng}(\mathsf{f}) \subseteq \mathrm{B}$ we denote the domain and range of f, respectively.

A two-player turn-based *arena* is a tuple $\mathcal{A} = \langle \mathrm{Ps}^0, \mathrm{Ps}^1, Mv \rangle$, with $\mathrm{Ps}^0 \cap \mathrm{Ps}^1 = \emptyset$ and $\mathrm{Ps} \triangleq \mathrm{Ps}^0 \cup \mathrm{Ps}^1$, such that $\langle \mathrm{Ps}, Mv \rangle$ is a finite directed graph. Ps^0 (*resp.*, Ps^1) is the set of positions of player 0 (*resp.*, 1) and $Mv \subseteq \mathrm{Ps} \times \mathrm{Ps}$ is a left-total relation describing all possible moves. A *path* in $\mathrm{V} \subseteq \mathrm{Ps}$ is a finite or infinite sequence $\pi \in \mathrm{Pth}(\mathrm{V})$ of positions in V compatible with the move relation, *i.e.*, $(\pi_i, \pi_{i+1}) \in Mv$, for all $i \in [0, |\pi| - 1[$. For a finite path π, with $\mathsf{lst}(\pi)$ we denote the last position of π. A positional *strategy* for player $\alpha \in \{0, 1\}$ on $\mathrm{V} \subseteq \mathrm{Ps}$ is a partial function $\sigma_\alpha \in \mathrm{Str}^\alpha(\mathrm{V}) \subseteq (\mathrm{V} \cap \mathrm{Ps}^\alpha) \rightharpoonup \mathrm{V}$, mapping each α-position $v \in \mathsf{dom}(\sigma_\alpha)$ to position $\sigma_\alpha(v)$ compatible with the move relation, *i.e.*, $(v, \sigma_\alpha(v)) \in Mv$. With $\mathrm{Str}^\alpha(\mathrm{V})$ we denote the set of all α-strategies on V. A *play* in $\mathrm{V} \subseteq \mathrm{Ps}$ from a position $v \in \mathrm{V}$ *w.r.t.* a pair of strategies $(\sigma_0, \sigma_1) \in \mathrm{Str}^0(\mathrm{V}) \times \mathrm{Str}^1(\mathrm{V})$, called $((\sigma_0, \sigma_1), v)$-*play*, is a path $\pi \in \mathrm{Pth}(\mathrm{V})$ such that $\pi_0 = v$ and, for all $i \in [0, |\pi| - 1[$, if $\pi_i \in \mathrm{Ps}^0$ then $\pi_{i+1} = \sigma^0(\pi_i)$ else $\pi_{i+1} = \sigma^1(\pi_i)$. The *play function* $\mathsf{play} : (\mathrm{Str}^0(\mathrm{V}) \times \mathrm{Str}^1(\mathrm{V})) \times \mathrm{V} \to \mathrm{Pth}(\mathrm{V})$ returns, for each position $v \in \mathrm{V}$ and pair of strategies $(\sigma_0, \sigma_1) \in \mathrm{Str}^0(\mathrm{V}) \times \mathrm{Str}^1(\mathrm{V})$, the maximal $((\sigma_0, \sigma_1), v)$-play $\mathsf{play}((\sigma^0, \sigma^1), v)$.

A *parity game* is a tuple $\mathcal{D} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle$, where $\mathcal{A}$ is an arena, $\mathrm{Pr} \subset \mathbb{N}$ is a finite set of priorities, and $\mathsf{pr} : \mathrm{Ps} \to \mathrm{Pr}$ is a *priority function* assigning a priority to each position. The priority function can be naturally extended to

games and paths as follows: $\mathsf{pr}(\mathfrak{D}) \triangleq \mathsf{max}_{v \in \mathrm{Ps}}\, \mathsf{pr}(v)$; for a path $\pi \in \mathrm{Pth}$, we set $\mathsf{pr}(\pi) \triangleq \mathsf{max}_{i \in [0,|\pi|[}\, \mathsf{pr}(\pi_i)$, if π is finite, and $\mathsf{pr}(\pi) \triangleq \limsup_{i \in \mathbb{N}} \mathsf{pr}(\pi_i)$, otherwise. A set of positions $\mathrm{V} \subseteq \mathrm{Ps}$ is an *α-dominion*, with $\alpha \in \{0,1\}$, if there exists an α-strategy $\sigma_\alpha \in \mathrm{Str}^\alpha(\mathrm{V})$ such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}(\mathrm{V})$ and positions $v \in \mathrm{V}$, the induced play $\pi = \mathsf{play}((\sigma_{\mathrm{o}}, \sigma_{\mathrm{1}}), v)$ is infinite and $\mathsf{pr}(\pi) \equiv_2 \alpha$. In other words, σ_α only induces on V infinite plays whose maximal priority visited infinitely often has parity α. By $\mathfrak{D}\backslash \mathrm{V}$ we denote the maximal subgame of $\mathfrak{D}$ with set of positions Ps' contained in $\mathrm{Ps}\backslash \mathrm{V}$ and move relation Mv' equal to the restriction of Mv to Ps'.

The α-predecessor of V, in symbols $\mathsf{pre}^\alpha(\mathrm{V}) \triangleq \{v \in \mathrm{Ps}^\alpha : Mv(v) \cap \mathrm{V} \neq \emptyset\} \cup \{v \in \mathrm{Ps}^{\overline{\alpha}} : Mv(v) \subseteq \mathrm{V}\}$, collects the positions from which player α can force the game to reach some position in V with a single move. The α-attractor $\mathsf{atr}^\alpha(\mathrm{V})$ generalizes the notion of α-predecessor $\mathsf{pre}^\alpha(\mathrm{V})$ to an arbitrary number of moves. Thus, it corresponds to the least fix-point of that operator. When $\mathrm{V} = \mathsf{atr}^\alpha(\mathrm{V})$, we say that V is α-maximal. Intuitively, V is α-maximal if player α cannot force any position outside V to enter this set. For such a V, the set of positions of the subgame $\mathfrak{D} \setminus \mathrm{V}$ is precisely $\mathrm{Ps} \setminus \mathrm{V}$. Finally, the *$\alpha$-escape* of V, formally $\mathsf{esc}^\alpha(\mathrm{V}) \triangleq \mathsf{pre}^\alpha(\mathrm{Ps} \setminus \mathrm{V}) \cap \mathrm{V}$, contains the positions in V from which α can leave V in one move, while the dual notion of *α-interior*, defined as $\mathsf{int}^\alpha(\mathrm{V}) \triangleq (\mathrm{V} \cap \mathrm{Ps}^\alpha) \setminus \mathsf{esc}^\alpha(\mathrm{V})$, contains the α-positions from which α cannot escape with a single move.

3 A New Idea

A solution for a parity game $\mathfrak{D} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr}\rangle \in \mathrm{PG}$ over an arena $\mathcal{A} = \langle \mathrm{Ps}^{\mathrm{o}}, \mathrm{Ps}^{\mathrm{1}}, Mv\rangle$ can trivially be obtained by iteratively computing dominions of some player, namely sets of positions from which that player has a strategy to win the game. Once an α-dominion D for player $\alpha \in \{0,1\}$ is found, its α-attractor $\mathsf{atr}^\alpha_{\mathfrak{D}}(\mathrm{D})$ gives an α-maximal dominion containing D, *i.e.*, α cannot force any position outside D to enter this set. The subgame $\mathfrak{D} \setminus \mathsf{atr}^\alpha_{\mathfrak{D}}(\mathrm{D})$ can then be solved by iterating the process. Therefore, the crucial problem to address consists in computing a dominion for some player in the game. The difficulty here is that, in general, no unique priority exists which satisfies the winning condition for a player along all the plays inside the dominion. In fact, that value depends on the strategy chosen by the opponent. Our solution to this problem is to proceed in a bottom-up fashion, starting from a weaker notion of α-dominion, called *quasi α-dominion*. Then, we compose quasi α-dominions until we obtain an α-dominion. Intuitively, a quasi α-dominion is a set of positions on which player α has a strategy whose induced plays either remain in the set forever and are winning for α or can exit from it. This notion is formalised by the following definition.

Definition 1 (Quasi Dominion). *Let $\mathfrak{D} \in \mathrm{PG}$ be a game and $\alpha \in \{0,1\}$ a player. A non-empty set of positions $\mathrm{Q} \subseteq \mathrm{Ps}_{\mathfrak{D}}$ is a* quasi α-dominion *in $\mathfrak{D}$ if there exists an α-strategy $\sigma_\alpha \in \mathrm{Str}^\alpha_{\mathfrak{D}}(\mathrm{Q})$ such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{Q})$, with $\mathsf{int}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\alpha}})$, and positions $v \in \mathrm{Q}$, the induced play*

$\pi = \mathsf{play}_{\mathfrak{I}}((\sigma_0, \sigma_1), v)$ *satisfies* $\mathsf{pr}_{\mathfrak{I}}(\pi) \equiv_2 \alpha$*, if* π *is infinite, and* $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\alpha}}_{\mathfrak{I}}(\mathrm{Q})$*, otherwise.*

The additional requirement that the opponent strategies be defined on all interior positions discards those strategies in which the opponent deliberately chooses to forfeit the play by declining to take any move at some of its positions.

We say that a quasi α-dominion Q is α*-open* (*resp.*, α*-closed*) if $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{I}}(\mathrm{Q}) \neq \emptyset$ (*resp.*, $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{I}}(\mathrm{Q}) = \emptyset$). In other words, in a closed quasi α-dominion, player α has a strategy whose induced plays are all infinite and winning. Hence, when closed, a quasi α-dominion is a dominion for α in $\mathfrak{I}$. The set of pairs $(\mathrm{Q}, \alpha) \in 2^{\mathrm{Ps}_{\mathfrak{I}}} \times \{0, 1\}$, where Q is a quasi α-dominion, is denoted by $\mathrm{QD}_{\mathfrak{I}}$, and is partitioned into the sets $\mathrm{QD}^{-}_{\mathfrak{I}}$ and $\mathrm{QD}^{+}_{\mathfrak{I}}$ of open and closed quasi α-dominion pairs, respectively.

Note that quasi α-dominions are loosely related with the concept of snares, introduced in [21] and used for completely different purposes, namely to speed up the convergence of strategy improvement algorithms.

During the search for a dominion, we explore a suitable partial order, whose elements, called *states*, record information about the open quasi dominions computed so far. The search starts from the top element, where the quasi dominions are initialised to the sets of nodes with the same priority. At each step, a query is performed on the current state to extract a new quasi dominion, which is then used to compute a successor state, if it is open. If, on the other hand, it is closed, the search is over. Different query and successor operations can in principle be defined, even on the same partial order. However, such operations cannot be completely independent. To account for this intrinsic dependence, we introduce a compatibility relation between states and quasi dominions that can be extracted by the query operation. Such a relation also forms the domain of the successor function. The partial order together with the query and successor operations and the compatibility relation forms what we call a *dominion space*.

Definition 2 (Dominion Space). *A* dominion space *for a game* $\mathfrak{I} \in \mathrm{PG}$ *is a tuple* $\mathcal{D} \triangleq \langle \mathfrak{I}, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where* (1) $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$ *is a well-founded partial order* w.r.t. $\prec \subset \mathrm{S} \times \mathrm{S}$ *with distinguished element* $\top \in \mathrm{S}$*,* (2) $\succ \subseteq \mathcal{S} \times \mathrm{QD}^{-}_{\mathfrak{I}}$ *is the* compatibility relation*,* (3) $\Re : \mathrm{S} \to \mathrm{QD}_{\mathfrak{I}}$ *is the* query function *mapping each element* $s \in \mathrm{S}$ *to a quasi dominion pair* $(\mathrm{Q}, \alpha) \triangleq \Re(s) \in \mathrm{QD}_{\mathfrak{I}}$ *such that, if* $(\mathrm{Q}, \alpha) \in \mathrm{QD}^{-}_{\mathfrak{I}}$ *then* $s \succ (\mathrm{Q}, \alpha)$*, and* (4) $\downarrow : \succ \to \mathrm{S}$ *is the* successor function *mapping each pair* $(s, (\mathrm{Q}, \alpha)) \in \succ$ *to the element* $s^\star \triangleq s\downarrow(\mathrm{Q}, \alpha) \in \mathrm{S}$ *with* $s^\star \prec s$*.*

The *depth* of a dominion space $\mathcal{D}$ is the length of the longest chain in the underlying partial order $\mathcal{S}$ starting from $\top$. Instead, by *execution depth* of $\mathcal{D}$ we mean the length of the longest chain induced by the successor function $\downarrow$. Obviously, the execution depth is always bound by the depth.

Different dominion spaces can be associated to the same game. Therefore, in the rest of this section, we shall simply assume a function Γ mapping every game $\mathfrak{I}$ to a dominion space $\Gamma(\mathfrak{I})$. Given the top element of $\mathcal{D} = \Gamma(\mathfrak{I})$, Algorithm 1 searches for a dominion of either one of the two players by querying the current state s for a region pair (Q, α). If this is closed in $\mathfrak{I}$, it is returned as an α-dominion. Otherwise, a successor state $s\downarrow_{\mathcal{D}}(\mathrm{Q}, \alpha)$ is computed and the search proceeds

recursively from it. Clearly, since the partial order is well-founded, termination of the $\mathsf{src}_{\mathcal{D}}$ procedure is guaranteed. The total number of recursive calls is, therefore, the execution depth $\mathsf{d}_{\mathcal{D}}(n, e, k)$ of the dominion space $\mathcal{D}$, where n, e, and k are the number of positions, moves, and priorities, respectively. Hence, $\mathsf{src}_{\mathcal{D}}$ runs in time $\mathrm{O}(\mathsf{d}_{\mathcal{D}}(n, e, k) \cdot (\mathsf{T}_{\Re}(n, e) + \mathsf{T}_{\downarrow}(n, e))$, where $\mathsf{T}_{\Re}(n, e)$ and $\mathsf{T}_{\downarrow}(n, e)$ denote the time needed by the query and successor functions, respectively. Thus, the total time to solve a game is $\mathrm{O}(e + n \cdot \mathsf{d}_{\mathcal{D}}(n, e, k) \cdot (\mathsf{T}_{\Re}(n, e) + \mathsf{T}_{\downarrow}(n, e)))$. Since the query and successor functions of the dominion space considered in the rest of the paper can be computed in linear time *w.r.t.* both n and e, the whole procedure terminates in time $\mathrm{O}(n \cdot (n + e) \cdot \mathsf{d}_{\mathcal{D}}(n, e, k))$. As to the space requirements, observe that $\mathsf{src}_{\mathcal{D}}$ is a tail recursive algorithm. Hence, the upper bound on memory only depends on the space needed to encode the states of a dominion space, namely $\mathrm{O}(\log \|\mathcal{D}\|)$, where $\|\mathcal{D}\|$ is the size of the partial order $\mathcal{S}$ associated with $\mathcal{D}$.

Soundness of the approach follows from the observation that quasi α-dominions closed in the entire game are winning for player α and so are their α-attractors. *Completeness*, instead, is ensured by the nature of dominion spaces. Indeed, algorithm $\mathsf{src}_{\mathcal{D}}$ always terminates by well-foundedness of the underlying partial order and, when it eventually does, a dominion for some player is returned. Therefore, the correctness of the algorithm reduces to proving the existence of a suitable dominion space, which is the subject of the next section.

Algorithm 1: The Searcher.

signature $\mathsf{src}_{\Gamma} : \mathrm{PG} \to_{\mathfrak{D}} \mathrm{QD}^{+}_{\mathfrak{D}}$
function $\mathsf{src}_{\Gamma}(\mathfrak{D})$
1 **return** $\mathsf{src}_{\Gamma(\mathfrak{D})}(\top_{\Gamma(\mathfrak{D})})$
signature $\mathsf{src}_{\mathcal{D}} : \mathrm{S}_{\mathcal{D}} \to \mathrm{QD}^{+}_{\mathfrak{D}_{\mathcal{D}}}$
function $\mathsf{src}_{\mathcal{D}}(s)$
1 $(\mathrm{Q}, \alpha) \leftarrow \Re_{\mathcal{D}}(s)$
2 **if** $(\mathrm{Q}, \alpha) \in \mathrm{QD}^{+}_{\mathfrak{D}_{\mathcal{D}}}$ **then**
3 **return** (Q, α)
else
4 **return** $\mathsf{src}_{\mathcal{D}}(s{\downarrow_{\mathcal{D}}}(\mathrm{Q}, \alpha))$

4 Priority Promotion

In order to compute dominions, we shall consider a restricted form of quasi dominions that constrains the escape set to have the maximal priority in the game. Such quasi dominions are called *regions*.

Definition 3 (Region). *A quasi α-dominion* R *is an α-region if* $\mathsf{pr}(\mathfrak{D}) \equiv_2 \alpha$ *and all the positions in* $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R})$ *have priority* $\mathsf{pr}(\mathfrak{D})$, i.e. $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathfrak{D}}(\mathsf{pr}(\mathfrak{D}))$.

As a consequence of the above definition, if the opponent $\overline{\alpha}$ can escape from an α-region, it must visit a position with the highest priority in the region, which is of parity α. Similarly to the case of quasi dominions, we shall denote with $\mathrm{Rg}_{\mathfrak{D}}$ the set of region pairs in $\mathfrak{D}$ and with $\mathrm{Rg}^{-}_{\mathfrak{D}}$ and $\mathrm{Rg}^{+}_{\mathfrak{D}}$ the sets of open and closed region pairs, respectively. A closed α-region is clearly an α-dominion.

At this point, we have all the tools to explain the crucial steps underlying the search procedure. Open regions are not winning, as the opponent can force plays exiting from them. Therefore, in order to build a dominion starting from open regions, we look for a suitable sequence of regions that can be merged together

until a closed one is found. Obviously, the merging operation needs to be applied only to regions belonging to the same player, in such a way that the resulting set of position is still a region of that player. To this end, a mechanism is proposed, where an α-region R in some game $\mathcal{D}$ and an α-dominion D in a subgame of $\mathcal{D}$ not containing R itself are merged together, if the only moves exiting from $\overline{\alpha}$-positions of D in the entire game lead to higher priority α-regions and R has the lowest priority among them. As we shall see, this ensures that the new region $\mathrm{R}^\star \triangleq \mathrm{R} \cup \mathrm{D}$ has the same associated priority as R. This merging operation, based on the following proposition, is called *promotion* of the lower region to the higher one.

Proposition 1 (Region Merging). *Let $\mathcal{D} \in \mathrm{PG}$ be a game, $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}}$ an α-region, and $\mathrm{D} \subseteq \mathrm{Ps}_{\mathcal{D}\setminus\mathrm{R}}$ an α-dominion in the subgame $\mathcal{D} \setminus \mathrm{R}$. Then, $\mathrm{R}^\star \triangleq \mathrm{R} \cup \mathrm{D}$ is an α-region in $\mathcal{D}$. Moreover, if both R and D are α-maximal in $\mathcal{D}$ and $\mathcal{D} \setminus \mathrm{R}$, respectively, then $\mathrm{R}^\star$ is α-maximal in $\mathcal{D}$ as well.*

Proof. Since R is an α-region, there is an α-strategy σ_{R} such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R})$, with $\mathsf{int}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \subseteq \mathsf{dom}(\sigma_{\overline{\alpha}})$, and positions $v \in \mathrm{R}$, the play induced by the two strategies is either winning for α or exits from R passing through a position of the escape set $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R})$, which must be one of the position of maximal priority in $\mathcal{D}$ and of parity α. Set D is, instead, an α-dominion in the game $\mathcal{D} \setminus \mathrm{R}$, therefore an α-strategy $\sigma_{\mathrm{D}} \in \mathrm{Str}_{\mathcal{D}\setminus\mathrm{R}}$ exists that is winning for α from every position in D, regardless of the strategy $\sigma'_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}_{\mathcal{D}\setminus\mathrm{R}}(\mathrm{D})$, with $\mathsf{int}^{\overline{\alpha}}_{\mathcal{D}\setminus\mathrm{R}}(\mathrm{D}) \subseteq \mathsf{dom}(\sigma'_{\overline{\alpha}})$, chosen by the opponent $\overline{\alpha}$. To show that $\mathrm{R}^\star$ is an α-region, it suffices to show that the following three conditions hold: *(i)* it is a quasi α-dominion; *(ii)* the maximal priority of $\mathcal{D}$ is of parity α; *(iii)* the escape set $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$ is contained in $\mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$.

Condition *(ii)* immediately follows from the assumption that R is an α-region in $\mathcal{D}$. To show that also Condition *(iii)* holds, we observe that, since D is an α-dominion in $\mathcal{D} \setminus \mathrm{R}$, the only possible moves exiting from $\overline{\alpha}$-positions of D in game $\mathcal{D}$ must lead to R, *i.e.*, $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{D}) \subseteq \mathrm{R}$. Hence, the only escaping positions of $\mathrm{R}^\star$, if any, must belong to R, *i.e.* $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star) \subseteq \mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R})$. Since R is an α-region in $\mathcal{D}$, it hods that $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$. By transitivity, we conclude that $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$.

Let us now consider Condition *(i)* and let the α-strategy $\sigma_{\mathrm{R}^\star} \triangleq \sigma_{\mathrm{R}} \cup \sigma_{\mathrm{D}}$ be defined as the union of the two strategies above. Note that, being D and R disjoint sets of positions, $\sigma_{\mathrm{R}^\star}$ is a well-defined strategy. We have to show that every path π compatible with $\sigma_{\mathrm{R}^\star}$ and starting from a position in $\mathrm{R}^\star$ is either winning for α or ends in a position of the escape set $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$.

First, observe that $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$ contains only those positions in the escaping set of R from which α cannot force to move into D, *i.e.* $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star) = \mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \setminus \mathsf{pre}^{\alpha}_{\mathcal{D}}(\mathrm{D})$.

Let now π be a play compatible with $\sigma_{\mathrm{R}^\star}$. If π is an infinite play, then it remains forever in $\mathrm{R}^\star$ and we have three possible cases. If π eventually remains forever in D, then it is clearly winning for α, since $\sigma_{\mathrm{R}^\star}$ coincides with σ_{D} on all the positions in D. Similarly, if π eventually remains forever in R, then it is also winning for α, as $\sigma_{\mathrm{R}^\star}$ coincides with σ_{R} on all the positions in R. If, on the other hand, π passes infinitely often through both R and D, it necessarily

visits infinitely often an escaping position in $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathfrak{D}}(\mathsf{pr}(\mathfrak{D}))$, which has the maximal priority in $\mathfrak{D}$ and is of parity α. Hence, the parity of the maximal priority visited infinitely often along π is α and π is winning for player α. Finally, if π is a finite play, then it must end at some escaping position of R from where α cannot force to move to a position still in $\mathrm{R}^\star$, *i.e.*, it must end in a position of the set $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R}) \setminus \mathsf{pre}^{\alpha}_{\mathfrak{D}}(\mathrm{D}) = \mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R}^\star)$. Therefore, $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{R}^\star)$. We can then conclude that $\mathrm{R}^\star$ also satisfies Condition *(i)*.

Let us now assume, by contradiction, that $\mathrm{R}^\star$ is not α-maximal. Then, there must be at least one position v belonging to $\mathsf{atr}^{\alpha}_{\mathfrak{D}}(\mathrm{R}^\star) \setminus \mathrm{R}^\star$, from which α can force entering $\mathrm{R}^\star$ in one move. Assume first that v is an α-position. Then there is a move from v leading either to R or to D. But this means that v belongs to either $\mathsf{atr}^{\alpha}_{\mathfrak{D}}(\mathrm{R}) \setminus \mathrm{R}$ or $\mathsf{atr}^{\alpha}_{\mathfrak{D}\setminus\mathrm{R}}(\mathrm{D}) \setminus \mathrm{D}$, contradicting α-maximality of those sets. If v is a $\overline{\alpha}$-position, instead, all its outgoing moves must lead to $\mathrm{R} \cup \mathrm{D}$. If all those moves lead to R, then $v \in \mathsf{atr}^{\alpha}_{\mathfrak{D}}(\mathrm{R}) \setminus \mathrm{R}$, contradicting α-maximality of R in $\mathfrak{D}$. If not, then in the subgame $\mathfrak{D} \setminus \mathrm{R}$, the remaining moves from v must all lead to D. But then, $v \in \mathsf{atr}^{\alpha}_{\mathfrak{D}\setminus\mathrm{R}}(\mathrm{D}) \setminus \mathrm{D}$, contradicting α-maximality of D in $\mathfrak{D} \setminus \mathrm{R}$.

During the search, we keep track of the computed regions by means of an auxiliary priority function $\mathsf{r} \in \Delta_{\mathfrak{D}} \triangleq \mathrm{Ps}_{\mathfrak{D}} \to \mathrm{Pr}_{\mathfrak{D}}$, called *region function*, which formalises the intuitive notion of priority of a region described above. Initially, the region function coincides with the priority function $\mathsf{pr}_{\mathfrak{D}}$ of the entire game $\mathfrak{D}$. Priorities are considered starting from the highest one. A region of the same parity $\alpha \in \{0, 1\}$ of the priority p under consideration is extracted from the region function, by collecting the set of positions $\mathsf{r}^{-1}(p)$. Then, its attractor $\mathrm{R} \triangleq \mathsf{atr}^{\alpha}_{\mathfrak{D}^\star}(\mathsf{r}^{-1}(p))$ is computed *w.r.t.* the subgame $\mathfrak{D}^\star$, which is derived from $\mathfrak{D}$ by removing the regions with priority higher than p. The resulting set forms an α-maximal set of positions from which the corresponding player can force a visit to positions with priority p. This first phase is called *region extension*. If the α-region R is open in $\mathfrak{D}^\star$, we proceed and process the next priority. In this case, we set the priority of the newly computed region to p. Otherwise, one of two situations may arise. Either R is closed in the whole game $\mathfrak{D}$ or the only $\overline{\alpha}$-moves exiting from R lead to higher regions of the same parity. In the former case, R is a α-dominion in the entire game and the search stops. In the latter case, R is only an α-dominion in the subgame $\mathfrak{D}^\star$, and a promotion of R to a higher region $\mathrm{R}^\natural$ can be performed, according to Proposition 1. The search, then, restarts from the priority of $\mathrm{R}^\natural$, after resetting to the original priorities in $\mathsf{pr}_{\mathfrak{D}}$ all the positions of the lower priority regions. The region $\mathrm{R}^\star$ resulting from the union of $\mathrm{R}^\natural$ and R will then be reprocessed and, possibly, extended in order to make it α-maximal. If R can be promoted to more than one region, the one with the lowest priority is chosen, so as to ensure the correctness of the merging operation. Due to the property of maximality, no $\overline{\alpha}$-moves from R to higher priority $\overline{\alpha}$-regions exist. Therefore, only regions of the same parity are considered in the promotion step. The correctness of region extension operation above, the remaining fundamental step in the proposed approach, is formalised by the following proposition.

Proposition 2 (Region Extension). *Let* $\mathfrak{D} \in \mathrm{PG}$ *be a game and* $\mathrm{R}^\star \subseteq \mathrm{Ps}_{\mathfrak{D}}$ *an* α*-region in* $\mathfrak{D}$*. Then,* $\mathrm{R} \triangleq \mathsf{atr}^{\alpha}_{\mathfrak{D}}(\mathrm{R}^\star)$ *is an* α*-maximal* α*-region in* $\mathfrak{D}$*.*

Proof. Since $\mathrm{R}^\star$ is an α-region in $\mathcal{D}$, then the maximal priority in $\mathcal{D}$ is of parity α and $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$. Hence, any position v in $\mathcal{D}$ must have priority $\mathsf{pr}_{\mathcal{D}}(v) \leq \mathsf{pr}(\mathcal{D})$. Player α can force entering $\mathrm{R}^\star$ from every position in $\mathsf{atr}^{\alpha}_{\mathcal{D}}(\mathrm{R}^\star)\setminus\mathrm{R}^\star$, with a finite number of moves. Moreover, $\mathrm{R}^\star$ is a quasi α-dominion and the priorities of the positions in $\mathrm{Ps}_{\mathcal{D}} \setminus \mathrm{R}^\star$ are lower than or equal to $\mathsf{pr}(\mathcal{D}) \equiv_2 \alpha$. Hence, every play that remains in R forever either eventually remains forever in $\mathrm{R}^\star$ and is winning for α, or passes infinitely often through $\mathrm{R}^\star$ and $\mathsf{atr}^{\alpha}_{\mathcal{D}}(\mathrm{R}^\star) \setminus \mathrm{R}^\star$. In the latter case, that path must visit infinitely often a position in $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$ that has the maximal priority in $\mathcal{D}$ and has parity α. Hence, the play is winning for α. If, on the other hand, $\overline{\alpha}$ can force a play to exit from R, it can do so only by visiting some position in $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$. In other words, $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \subseteq \mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$. In either case, we conclude that R is an α-region in $\mathcal{D}$. Finally, being R the result of an α-attractor, it is clearly α-maximal.

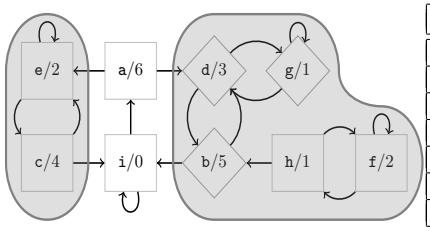

Fig. 1: Running example.

	1	2	3	4	5	6	7
6	a↓	···	···	···	···	a,b,d,g,i↓	···
5	b,f,h↓	···	···	b,d,f,g,h↓	···		
4	c↓	c,e↓	···	c↓	c,e↓	c↓	c,e,f,h↑$_6$
3	d↓	d↓	d,g↑$_5$				
2	e↑$_4$			e↑$_4$		e, f, h↑$_4$	
1		g↑$_3$					
0					i↑$_6$		

Table 1: PP simulation.

Figure 1 and Table 1 illustrate the search procedure on an example game, where diamond shaped positions belong to player 0 and square shaped ones to the opponent 1. Player 0 wins from every position, hence the 0-region containing all the positions is a 0-dominion in this case. Each cell of the table contains a computed region. A downward arrow denotes a region that is open in the subgame where it is computed, while an upward arrow means that the region gets to be promoted to the priority in the subscript. The index of each row corresponds to the priority of the region. Following the idea sketched above, the first region obtained is the single-position 0-region $\{\mathtt{a}\}$, which is open because of the two moves leading to `d` and `e`. At priority 5, the open 1-region $\{\mathtt{b}, \mathtt{f}, \mathtt{h}\}$ is formed by attracting both `f` and `h` to `b`, which is open in the subgame where $\{\mathtt{a}\}$ is removed. Similarly, the 0-region $\{\mathtt{c}\}$ at priority 4 and the 1-region $\{\mathtt{d}\}$ at priority 3 are open, once removed $\{\mathtt{a}, \mathtt{b}, \mathtt{f}, \mathtt{h}\}$ and $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{f}, \mathtt{h}\}$, respectively, from the game. At priority 2, the 0-region $\{\mathtt{e}\}$ is closed in the corresponding subgame. However, it is not closed in the whole game, since it has a move leading to `c`, *i.e.*, to region 4. A promotion of $\{\mathtt{e}\}$ to 4 is then performed, resulting in the new 0-region $\{\mathtt{c}, \mathtt{e}\}$. The search resumes at the corresponding priority and, after computing the extension of such a region via the attractor, we obtain that it is still open in the corresponding subgame. Consequently, the 1-region of priority 3 is recomputed and, then, priority 1 is processed to build the 1-region $\{\mathtt{g}\}$. The latter is closed in the associated subgame, but not in the original game, because of a move leading to position `d`. Hence, another promotion is performed, leading to closed

region in Row 3 and Column 3, which in turn triggers a promotion to 5. Observe that every time a promotion to a higher region is performed, all positions of the regions at lower priorities are reset to their original priorities. The iteration of the region forming and promotion steps proceeds until the configuration in Column 7 is reached. Here only two 0-regions are present: the open region 6 containing $\{\mathtt{a}, \mathtt{b}, \mathtt{d}, \mathtt{g}, \mathtt{i}\}$ and the closed region 4 containing $\{\mathtt{c}, \mathtt{e}, \mathtt{f}, \mathtt{h}\}$. The second one has a move leading to the first one, hence, it is promoted to its priority. This last operation forms a 0-region containing all the positions of the game. It is obviously closed in the whole game and is, therefore, a 0-dominion.

Note that, the positions in 0-region $\{\mathtt{c}, \mathtt{e}\}$ are reset to their initial priorities, when 1-region $\{\mathtt{d}, \mathtt{g}\}$ in Column 3 is promoted to 5. Similarly, when 0-region $\{\mathtt{i}\}$ in Column 5 is promoted to 6, the priorities of the positions in both regions $\{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and $\{\mathtt{c}, \mathtt{e}\}$, highlighted by the grey areas, are reset. This is actually necessary for correctness, at least in general. In fact, if region $\{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ were not reset, the promotion of $\{\mathtt{i}\}$ to 6, which also attracts $\mathtt{b}$, $\mathtt{d}$, and $\mathtt{g}$, would leave $\{\mathtt{f}, \mathtt{h}\}$ as a 1-region of priority 5. However, according to Definition 3, this is not a 1-region. Even worse, it would also be considered a closed 1-region in the entire game, without being a 1-dominion, since it is actually an open 0-region. This shows that, in principle, promotions to an higher priority require the reset of previously built regions of lower priorities.

In the rest of this section, we shall formalise the intuitive idea described above. The necessary conditions under which promotion operations can be applied are also stated. Finally, query and successor algorithms are provided, which ensure that the necessary conditions are easy to check and always met when promotions are performed.

The* PP *Dominion Space. In order to define the dominion space induced by the *priority-promotion mechanism* (PP, for short), we need to introduce some additional notation. Given a priority function $\mathsf{r} \in \mathit{\Delta}_{\mathcal{D}}$ and a priority $p \in \mathrm{Pr}$, we denote by $\mathsf{r}^{(\geq p)}$ (*resp.*, $\mathsf{r}^{(>p)}$ and $\mathsf{r}^{(<p)}$) the function obtained by restricting the domain of r to the positions with priority greater than or equal to p (*resp.*, greater than and lower than p). Formally, $\mathsf{r}^{(\geq p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) \geq p\}$, $\mathsf{r}^{(>p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) > p\}$, and $\mathsf{r}^{(<p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) < p\}$. By $\mathcal{D}_{\mathsf{r}}^{\leq p}$ we denote the largest subgame contained in the structure $\mathcal{D} \setminus \mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$, which is obtained by removing from $\mathcal{D}$ all the positions in the domain of $\mathsf{r}^{(>p)}$.

A priority function $\mathsf{r} \in \mathbb{R}_{\mathcal{D}} \subseteq \mathit{\Delta}_{\mathcal{D}}$ in $\mathcal{D}$ is a *region function* iff, for all priorities $q \in \mathsf{rng}(\mathsf{r})$ with $\alpha \triangleq q \bmod 2$, it holds that $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}_{\mathsf{r}}^{\leq q}}$ is an α-region in the subgame $\mathcal{D}_{\mathsf{r}}^{\leq q}$, if non-empty. In addition, we say that r is *maximal* above $p \in \mathrm{Pr}$ iff, for all $q \in \mathsf{rng}(\mathsf{r})$ with $q > p$, we have that $\mathsf{r}^{-1}(q)$ is α-maximal in $\mathcal{D}_{\mathsf{r}}^{\leq q}$ with $\alpha \triangleq q \bmod 2$.

To account for the current status of the search of a dominion, the states s of the corresponding dominion space need to contain the current region function r and the current priority p reached by the search in $\mathcal{D}$. To each of such states $s \triangleq (\mathsf{r}, p)$, we then associate the *subgame at* s defined as $\mathcal{D}_s \triangleq \mathcal{D}_{\mathsf{r}}^{\leq p}$, representing the portion of the original game that still has to be processed.

We can now formally define the *Priority Promotion* dominion space, by characterising the corresponding state space and compatibility relation. Moreover, algorithms for the query and successor functions of that space are provided.

Definition 4 (State Space). *A* state space *is a tuple* $\mathcal{S}_\partial \triangleq \langle \mathrm{S}_\partial, \top_\partial, \prec_\partial \rangle$*, where its components are defined as prescribed in the following:*

1. $\mathrm{S}_\partial \subseteq \mathbb{R}_\partial \times \mathrm{Pr}_\partial$ *is the set of all pairs* $s \triangleq (\mathsf{r}, p)$*, called* states*, composed of a region function* $\mathsf{r} \in \mathbb{R}_\partial$ *and a priority* $p \in \mathrm{Pr}_\partial$ *such that* (a) r *is maximal above* p*,* (b) $p \in \mathsf{rng}(\mathsf{r})$*, and* (c) $\mathsf{r}^{(<p)} \subseteq \mathsf{pr}_\partial{}^{(<p)}$*;*
2. $\top_\partial \triangleq (\mathsf{pr}_\partial, \mathsf{pr}(\partial))$*;*
3. *for any two states* $s_1 \triangleq (\mathsf{r}_1, p_1), s_2 \triangleq (\mathsf{r}_2, p_2) \in \mathrm{S}_\partial$*, it holds that* $s_1 \prec_\partial s_2$ *iff either* (a) *there exists a priority* $q \in \mathsf{rng}(\mathsf{r}_1)$ *with* $q \geq p_1$ *such that* (a.i) $\mathsf{r}_1{}^{(>q)} = \mathsf{r}_2{}^{(>q)}$ *and* (a.ii) $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$*, or* (b) *both* (b.i) $\mathsf{r}_1 = \mathsf{r}_2$ *and* (b.ii) $p_1 < p_2$ *hold.*

The state space specifies the configurations in which the priority promotion procedure can reside and the relative order that the successor function must satisfy. In particular, for a given state $s \triangleq (\mathsf{r}, p)$, every region $\mathsf{r}^{-1}(q)$, with priority $q > p$, recorded in the region function r has to be α-maximal, where $\alpha = q \bmod 2$. This implies that $\mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\partial_{\mathsf{r}}^{\leq q}}$. Moreover, the current priority p of the state must be the priority of an actual region in r. Finally, all the regions recorded in r at any priority q lower than p must contain positions that have the same priority q in the original priority function pr_∂ of the game. As far as the order is concerned, a state s_1 is strictly smaller than another state s_2 if either there is a region recorded in s_1 at some higher priority that strictly contains the corresponding one in s_2 and all regions above are equal in the two states, or state s_1 is currently processing a lower priority than the one of s_2.

Algorithm 2: Query Function.

```
signature ℜ_∂ : S_∂ → 2^{Ps_∂} × {0,1}
function ℜ_∂(s)
    let (r, p) = s in
1       α ← p mod 2
2       R ← atr^α_{∂_s}(r^{-1}(p))
3   return (R, α)
```

At this point, we can determine the regions that are compatible with a given state. They are the only ones that the query function is allowed to return and that can then be used by the successor function to make the search progress in the dominion space. Intuitively, a region pair (R, α) is compatible with a state $s \triangleq (\mathsf{r}, p)$ if it is an α-region in the current subgame ∂_s. Moreover, if such region is α-open in that game, it has to be α-maximal, and it has to necessarily contain the current region $\mathsf{r}^{-1}(p)$ of priority p in r. These three accessory properties ensure that the successor function is always able to cast R inside the current region function r and obtain a new state.

Definition 5 (Compatibility Relation). *An open quasi dominion pair* $(\mathrm{R}, \alpha) \in \mathrm{QD}^-_\partial$ *is* compatible *with a state* $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_\partial$*, in symbols* $s \succ_\partial (\mathrm{R}, \alpha)$*, iff* (1) $(\mathrm{R}, \alpha) \in \mathrm{Rg}_{\partial_s}$ *and* (2) *if* R *is* α*-open in* ∂_s *then* (2.a) R *is* α*-maximal in* ∂_s *and* (2.b) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$*.*

Algorithm 2 provides a possible implementation for the query function compatible with the priority-promotion mechanism. Let $s \triangleq (\mathsf{r}, p)$ be the current state. Line 1 simply computes the parity α of the priority to process in that state. Line 2, instead, computes in game $\mathcal{D}_s$ the attractor *w.r.t.* player α of the region contained in r at the current priority p. The resulting set R is, according to Proposition 2, an α-maximal α-region in $\mathcal{D}_s$ containing $\mathsf{r}^{-1}(p)$.

Before continuing with the description of the implementation of the successor function, we need to introduce the notion of *best escape priority* for player $\overline{\alpha}$ *w.r.t.* an α-region R of the subgame $\mathcal{D}_s$ and a region function r in the whole game $\mathcal{D}$. Informally, such a value represents the best priority associated with an α-region contained in r and reachable by $\overline{\alpha}$ when escaping from R. To formalise this concept, let $I \triangleq Mv_{\mathcal{D}} \cap ((\mathrm{R} \cap \mathrm{Ps}_{\mathcal{D}}^{\overline{\alpha}}) \times (\mathsf{dom}(\mathsf{r}) \setminus \mathrm{R}))$ be the *interface relation* between R and r, *i.e.*, the set of $\overline{\alpha}$-moves exiting from R and reaching some position within a region recorded in r. Then, $\mathsf{bep}_{\mathcal{D}}^{\overline{\alpha}}(\mathrm{R}, \mathsf{r})$ is set to the minimal priority among those regions containing positions reachable by a move in I. Formally, $\mathsf{bep}_{\mathcal{D}}^{\overline{\alpha}}(\mathrm{R}, \mathsf{r}) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r}{\restriction}\mathsf{rng}(I)))$. Note that, if R is a closed α-region in $\mathcal{D}_s$, then $\mathsf{bep}_{\mathcal{D}}^{\overline{\alpha}}(\mathrm{R}, \mathsf{r})$ is necessarily of parity α and greater than the priority p of R. This property immediately follows from the maximality of r above p in any state of the dominion space. Indeed, no move of an $\overline{\alpha}$-position can lead to a $\overline{\alpha}$-maximal $\overline{\alpha}$-region. For instance, in the example of Figure 1, for 0-region $\mathrm{R} = \{\mathsf{e}, \mathsf{f}, \mathsf{h}\}$ with priority equal to 2 in column 6, we have that $I = \{(\mathsf{e}, \mathsf{c}), (\mathsf{h}, \mathsf{b})\}$ and $\mathsf{r}{\restriction}\mathsf{rng}(I) = \{(\mathsf{c}, 4), (\mathsf{b}, 6)\}$. Hence, $\mathsf{bep}_{\mathcal{D}}^{1}(\mathrm{R}, \mathsf{r}) = 4$.

In the following, to reset the priority of some the positions in the game, after a promotion of a given region is performed, we define the *completing operator* $\uplus$ that, taken a partial function $\mathsf{f} : \mathrm{A} \rightharpoonup \mathrm{B}$ and a total function $\mathsf{g} : \mathrm{A} \to \mathrm{B}$, returns the total function $\mathsf{g} \uplus \mathsf{f} \triangleq (\mathsf{g} \setminus \mathsf{dom}(\mathsf{f})) \cup \mathsf{f} : \mathrm{A} \to \mathrm{B}$. The result is equal to f on its domain and assumes the same values of g on the remaining part of the set A.

Algorithm 3 implements the successor function informally described at the beginning of the section. Given the current state s and a compatible region pair (R, α) open in the whole game as inputs, it produces a successor state $s^\star \triangleq (\mathsf{r}^\star, p^\star)$ in the dominion space. It first checks whether R is open also in the subgame $\mathcal{D}_s$ (Line 1). If this is the case, it assigns priority p to region R and stores it in the new region function $\mathsf{r}^\star$ (Line 2). The new current priority $p^\star$ is, then, computed as the highest priority lower than p in $\mathsf{r}^\star$ (Line 3). If, on the other hand, R is closed in $\mathcal{D}_s$, a promotion merging R with some other α-region contained in r is required. The next priority $p^\star$ is set to the bep of R for player $\overline{\alpha}$ in the entire game $\mathcal{D}$ *w.r.t.* r (Line 4). Region R is, then, promoted to priority $p^\star$ and

Algorithm 3: Successor Function.

```
   signature ↓_D : ≻_D → Δ_D × Pr_D
   function s↓_D (R, α)
       let (r, p) = s in
1          if (R, α) ∈ Rg⁻_{D_s} then
2              r* ← r[R ↦ p]
3              p* ← max(rng(r*^(<p))))
           else
4              p* ← bep^ᾱ_D(R, r)
5              r* ← pr_D ⊎ r^(≥p*)[R ↦ p*]
6      return (r*, p*)
```

all the priorities below $p^\star$ in the current region function r are reset (Line 5). The correctness of this last operation follows from Proposition 1.

As already observed in Section 3, a dominion space, together with Algorithm 1, provides a sound and complete solution procedure. The following theorem states that the priority-promotion mechanism presented above is indeed a dominion space. The proof will be provided in the extended version of the paper.

Theorem 1 (Dominion Space). *For a game $\mathcal{D}$, the* structure $\mathcal{D}_\mathcal{D} \triangleq \langle \mathcal{D}, \mathcal{S}_\mathcal{D}, \succ_\mathcal{D}, \Re_\mathcal{D}, \downarrow_\mathcal{D} \rangle$*, where $\mathcal{S}_\mathcal{D}$ is given in Definition 4, $\succ_\mathcal{D}$ is the relation of Definition 5, and $\Re_\mathcal{D}$ and $\downarrow_\mathcal{D}$ are the functions computed by Algorithms 2 and 3 is a dominion space.*

Complexity of* PP *Dominion Space. To conclude, we estimate the size and depth of dominion space $\mathcal{R}_\mathcal{D}$. This provides upper bounds on both the time and space needed by the search procedure $\mathsf{src}_{\mathcal{R}_\mathcal{D}}$ computing dominions. By looking at the definition of state space $\mathcal{S}_\mathcal{D}$, it is immediate to see that, for a game $\mathcal{D}$ with n positions and k priorities, the number of states is bounded by k^n. Indeed, there are at most k^n functions $\mathsf{r} : \mathrm{Ps}_\mathcal{D} \to \mathrm{Pr}_\mathcal{D}$ from positions to priorities that can be used as region function of a state. Note that the associated current priority is uniquely determined by the content of the region function. Measuring the depth is a little trickier. A coarse bound can be obtained by observing that there is an homomorphism from $\mathcal{S}_\mathcal{D}$ to the well-founded partial order, in which the region function r of a state is replaced by a partial function $\mathsf{f} : \mathrm{Pr}_\mathcal{D} \rightharpoonup [1, n]$ with the following properties: it assigns to each priority $p \in \mathsf{rng}(\mathsf{r})$ the size $\mathsf{f}(p)$ of the associated region $\mathsf{r}^{-1}(p)$. The order $(\mathsf{f}_1, p_1) \prec (\mathsf{f}_2, p_2)$ between two pairs is derived from the one on the states, by replacing $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$ with $\mathsf{f}_2(q) < \mathsf{f}_1(q)$. This homomorphism ensures that every chain in $\mathcal{S}_\mathcal{D}$ corresponds to a chain in the new partial order. Moreover, there are exactly $\binom{n+k}{k}$ partial functions f such that $\sum_{p \in \mathsf{dom}(\mathsf{f})} \mathsf{f}(p) \leq n$. Consequently, every chain cannot be longer than $\binom{n+k}{k} \leq \left(\mathsf{e}(\frac{n}{k} + 1)\right)^k$, where e is the Euler constant. By further exploiting the structure of the space, one can obtain a recurrence relation expressing a slightly better upper bound, whose explicit solution is $3 \cdot \sum_{i=0}^{k-2} \binom{n-2}{i}$. Then, by applying a standard approximation via geometric series based on the inequality $\left(\frac{h-i}{n-h+i}\right)^i \leq \binom{m}{h-i} / \binom{m}{h} \leq \left(\frac{h}{n-h+1}\right)^i$, we derive the asymptotic bound stated by the following theorem. A formal account of the recurrence relation will be provided in the extended version of this article.

Theorem 2 (Size & Depth Upper Bounds). *The size of a* PP *dominion space $\mathcal{R}$ with $n \in \mathbb{N}_+$ positions and $k \in [1, n]$ priorities is bounded by k^n. Moreover, if $2 \leq k$, its depth is bounded by $3 \cdot \sum_{i=0}^{k-2} \binom{n-2}{i}$, which is less than $3\frac{n-k+1}{n-2k+3} \left(\mathsf{e}\frac{n-2}{k-2}\right)^{k-2}$, if $k < n/2$, and less than $3(2^{n-2} - c(\frac{n-2}{k-2})^{k-2})$, for a constant $c > 0$, otherwise.*

Unfortunately, due to the reset operations performed after each promotion, an exponential worst-case can actually be built. Indeed, consider the game $\mathcal{D}_{m,h}$

having all positions ruled by player 0 and containing h chains of length $2m+1$ that converge into a single position of priority 0 with a self loop. The i-th chain has a head of priority $4h-i$ and a body composed of m blocks of two positions having priority $2i-1$ and $2i$, respectively. The first position in each block also has a self loop. An instance of this game with $m=2$ and $h=4$ is depicted in Figure 2. The labels of the positions correspond to the associated priorities and the highlighted area at the bottom of the figure groups together the last blocks of the chains. Intuitively, the execution depth of the PP dominion space for this game is exponential, since the consecutive promotion operations performed on each chain can simulate the increments of a counter up to m. Also, the priorities are chosen in such a way that, when the i-th counter is incremented, all the j-th counters with $j \in]i, h]$ are reset. Therefore, the whole game simulates a counter with h digits taking values from 0 to m. Hence, the overall number of performed promotions is $(m+1)^h$. The search procedure on $\partial_{2,4}$ starts by building the four open 1-regions $\{15\}$, $\{13\}$, $\{11\}$, and $\{9\}$ and the open 0-region $\{8', 7'', 8''\}$, where we use apices to distinguish different positions with the same priority. This state represents the configuration of the counter, where all four digits are set to 0. The closed 1-region $\{7'\}$ is then found and promoted to 9. Consequently, the previously computed 0-region with priority 8 is reset and the new region is maximised to obtain the open 1-region $\{9, 7', 8'\}$. Now, the counter is set to 0001. After that, the open 0-region $\{8''\}$ and the closed 1-region $\{7''\}$ are computed. The latter one is promoted to 9 and maximised to attract position $8''$. This completes the 1-region containing the entire chain ending in 9. The value of the counter is now 0002. At this point, immediately after the construction of the open 0-region $\{6', 5'', 6''\}$, the closed 1-region $\{5'\}$ is found, promoted to 11, and maximised to absorb position $6'$. Due to the promotion, the positions in the 1-region with priority 9 are reset to their original priority and all the work done to build it gets lost. This last operation represents the reset of the least significant digit of the counter, caused by the increment of the second one, *i.e.*, the counter displays 0010. Following similar steps, the process carries on until each chain is grouped in a single region. The corresponding state represents the configuration of the counter in which all digits are set to m. Thus, after an exponential number promotions, the closed 0-region $\{0\}$ is eventually obtained as solution.

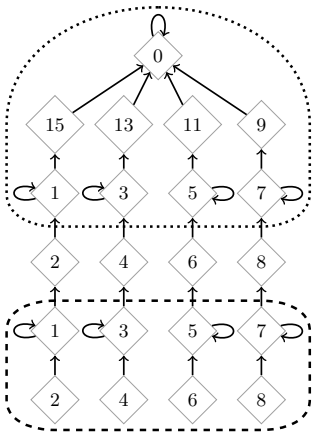

Fig. 2: The $\partial^{\mathrm{PP}}_{2,4}$ game.

Theorem 3 (Execution-Depth Lower Bounds). *For all numbers $h \in \mathbb{N}$, there exists a* PP *dominion space $\mathcal{R}_h$ with $k = 2h+1$ positions and priorities, whose execution depth is $3 \cdot 2^h - 2 = \Theta\big(2^{k/2}\big)$. Moreover, for all numbers $m \in \mathbb{N}_+$, there exists a* PP *dominion space $\mathcal{R}_{m,h}$ with $n = (2m+1) \cdot h + 1$ positions and $k = 3h+1$ priorities, whose execution depth is $((3m+1) \cdot (m+1)^h - 1)/m - 2 = \mathrm{O}\Big(\big(3n/(2(k-1))\big)^{k/3}\Big)$.*

Observe that, in the above theorem, we provide two different exponential lower bounds. The general one, with $k/3$ as exponent and a parametric base, is the result of the game $\eth_{m,h}$ described in the previous paragraph, where $k = 3h+1$. The other bound, instead, has a base fixed to 2, but the worse exponent $k/2$. We conjecture that the given upper bound could be improved to match the exponent $k/2$ of this lower bound. In this way, we would obtain an algorithm with an asymptotic behaviour comparable with the one exhibited by the small-progress measure procedure [32]. This study will be further pursued in the extended version of the article.

5 Experimental Evaluation

In order to assess the effectiveness of the proposed approach, the new technique described above has been implemented in the tool PGSolver [28], which collects implementations of several parity game solvers proposed in the literature. This software framework, implemented in OCaml, also provides a benchmarking tool, which can generate different forms of parity games. The available benchmarks divide into concrete problems and synthetic ones. The *concrete benchmarks* encode validity and verification problems for temporal logics. They consist in parity games resulting from encodings of the language inclusion problem between automata, specifically a non-deterministic Büchi automaton and a deterministic one, reachability problems, namely the Tower of Hanoi problem, and fairness verification problems, the Elevator problem (see [28]). The *synthetic benchmarks* divide into randomly generated games and various families corresponding to difficult cases (clique and ladder-like games) and worst cases of the solvers implemented in PGSolver. To fairly compare the different solution techniques used by the underlying algorithms, the solvers involved in the experiments have been isolated from the generic solver implemented in PGSolver, which exploits game transformation and decomposition techniques in the attempt to speed up the solution process. However, those optimisations can, in some cases, solve the game without even calling the selected algorithm, and, in other cases, the resulting overhead can even outweigh the solver time, making the comparison among solvers virtually worthless [28]. Experiments were also conducted with different optimisations enabled and the results exhibit the same pattern emerging in the following experimental evaluation.

The algorithms considered in the experimentation are the Zielonka algorithm *Rec* [54], its two dominion decomposition variants, *Dom* [33,34] and *Big* [48], the strategy improvement algorithm *Str* [52], and the one proposed in this article, PP. Small progress measure [32] is not included, since it could not solve any of the tested benchmarks within the available computational resources [3].

[3] Experiments were carried out on a 64-bit 3.1GHz Intel® quad-core machine, with i5-2400 processor and 8GB of RAM, running Ubuntu 12.04 with Linux kernel version 3.2.0. PGSolver was compiled with OCaml version 2.12.1.

Special Families. Table 2 displays the results of all the solvers involved on the benchmark families available in PGSolver. We only report on the biggest instances we could deal with, given the available computational resources[4]. The parameter Size refers to the number of positions in the games and the best performance are emphasised in bold. The first three rows consider the concrete verification problems mentioned above. On the Tower of Hanoi problem all the solvers perform reasonably well, except for *Str* due its high memory requirements. The Elevator problem proved to be very demanding in terms of memory for all the solvers, except for our new algorithm and *Dom*, which, however, could not solve it within the time limit of 10 minutes. Our solver performs extremely well on both this benchmark and on Language Inclusion, which could be solved only by *Rec* among the other solvers. On the worst case benchmarks, it performs quite well also on Ladder, Strategy Improvement, and Clique, which proved to be considerably difficult for all the other solvers. It was outperformed only on the last three ones: the Modelchecker, the Recursive Ladder, and Jurdziński games. Despite this fact, the new solver exhibit the most consistent behaviour overall on these benchmarks. Indeed, in all those benchmarks, the priority promotion algorithm requires no promotions regardless of the input parameters, except for the elevator problem, where it performs only two promotions.

Benchmark	Size	Dom	Big	Str	Rec	PP
Hanoi	6.3M	21.4	21.4	‡	17.4	**14.2**
Elevator	7.7M	†	‡	‡	‡	**43.3**
Lang. Incl.	5M	†	‡	‡	145.5	**21.1**
Ladder	4M	†	‡	‡	35.0	**17.1**
Str. Imp.	4.5M	81.0	82.8	†	71.0	**50.0**
Clique	8K	†	‡	†	†	**21.7**
MC. Lad.	7.5M	†	‡	‡	**4.3**	6.5
Rec. Lad.	50K	†	‡	**0.6**	‡	311.2
Jurdziński	40K	†	†	**188.2**	†	314.4

Table 2: Execution times in seconds on several benchmark families. Time out (†) is set to 600 seconds and memory out (‡) to 7.5Gb.

Random Games. Figure 3 compares the running times (left-hand side) and memory requirements (right-end side) of the new algorithm PP against *Rec* and *Str* on 2000 random games of size ranging from 5000 to 20000 positions and 2 outgoing moves per position. Interestingly, these random games proved to be quite challenging for all the considered solvers. We set a time-out to 180 seconds (3 minutes). Both *Dom* and *Big* perform quite poorly on those games, hitting the time-out already for very small instances, and we decided to leave them out of the picture. The behaviour of the solvers is typically highly variable even on games of the same size and priorities. To summarise the results, the average running time on clusters of games seemed the most appropriate choice in this case. Therefore, each point in the graph shows the average time over a cluster of

[4] The instances were generated with the following PGSolver commands: `towersofhanoi 13`, `elevatorgame 8`, `langincl 500 100`, `laddergame 4000000`, `stratimprgen -pg friedmannsubexp 1000`, `modelcheckerladder 2500000`, `cliquegame 8000`, `recursiveladder 10000`, and `jurdzinskigame 100 100`.

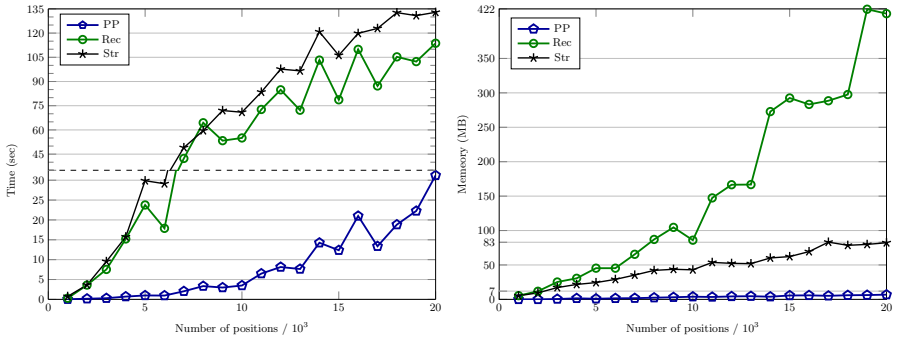

Fig. 3: Time and auxiliary memory on random games with 2 moves per position.

100 different games of the same size: for each size value n, we chose a number $k = n \cdot i/10$ of priorities, with $i \in [1, 10]$, and 10 random games were generated for each pair of n and k. The new algorithm perform significantly better than the others on those games. The right-hand side graph also shows that the theoretical improvement on the auxiliary memory requirements of the new algorithm has a considerable practical impact on memory consumption compared to the other solvers. We also experimented on random games with a higher number of moves per position. The resulting games turn out to be much easier to solve for all the solvers. This behaviour might depend on the specific random generator provided by PGSolver. However, those experiments still show better performance by the new algorithm *w.r.t.* the competitor ones. Due to the space constraints, the corresponding results will be reported in the extended version of the paper.

6 Discussion

We considered the problem of solving *Parity Games*, a special form of infinite-duration games over graphs having relevant applications in various branches of Theoretical Computer Science. We proposed a novel solution technique, based on a *priority-promotion mechanism*. Based on this approach, a new solution algorithm have been presented and studied. We gave proofs of its correctness and provided an accurate analysis of its time and space complexities.

As far as time complexity is concerned, an exponential upper bound in the number of priorities has been given. A lower bound for the worst-case was also presented in the form of a family of parity games on which the new technique exhibits an exponential behaviour. On the bright side, the new solution exhibits the best space complexity among the currently known algorithms for parity games. In fact, we showed that the maximal additional space needed to solve a parity game is linear in the number of positions, logarithmic in the number of priorities, and independent from the number of moves in the game. This is an important result, in particular considering that in practical applications we often need to deal with games having a very high number of positions, moves, and, in some cases, priorities. Therefore, low space requirements are essential for practical scalability.

To assess the effectiveness of the new approach, experiments were carried out against concrete and synthetic problems. We compared the new algorithm with the state-of-the-art solvers implemented in PGSolver. The results are very promising, showing that the proposed approach is extremely effective in practice, often substantially better than existing ones. This suggests that the new approach is worth pursuing further. Therefore, we are currently investigating new and clever priority-promotion policies that try to minimise the number of region resets after a priority promotion.

It would be interesting to investigate the applicability of the priority promotion approach to related problems, such as *prompt-parity games* [42] and similar conditions [12, 26, 27], and even in wider contexts like *mean-payoff games* [13, 16] and *energy games* [10, 11].

References

1. M. Agrawal, N. Kayal, and N. Saxena, "PRIMES is in P." *AM*, vol. 160, no. 2, pp. 781–793, 2004.
2. R. Alur, T. Henzinger, and O. Kupferman, "Alternating-Time Temporal Logic." *JACM*, vol. 49, no. 5, pp. 672–713, 2002.
3. K. Apt and E. Grädel, *Lectures in Game Theory for Computer Scientists.* Cambridge University Press, 2011.
4. M. Benerecetti, F. Mogavero, and A. Murano, "Substructure Temporal Logic." in *LICS'13*. IEEE Computer Society, 2013, pp. 368–377.
5. ——, "Reasoning About Substructures and Games." *TOCL*, vol. 16, no. 3, pp. 25:1–46, 2015.
6. D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer, "DAG-Width and Parity Games." in *STACS'06*, ser. LNCS 3884. Springer, 2006, pp. 524–536.
7. D. Berwanger and E. Grädel, "Games and Model Checking for Guarded Logics." in *LPAR'01*, ser. LNCS 2250. Springer, 2001, pp. 70–84.
8. ——, "Fixed-Point Logics and Solitaire Games." *TCS*, vol. 37, no. 6, pp. 675–694, 2004.
9. D. Berwanger, E. Grädel, L. Kaiser, and R. Rabinovich, "Entanglement and the Complexity of Directed Graphs." *TCS*, vol. 463, pp. 2–25, 2012.
10. K. Chatterjee and L. Doyen, "Energy Parity Games." *TCS*, vol. 458, pp. 49–60, 2012.
11. K. Chatterjee, L. Doyen, T. Henzinger, and J.-F. Raskin, "Generalized Mean-Payoff and Energy Games." in *FSTTCS'10*, ser. LIPIcs 8. Leibniz-Zentrum fuer Informatik, 2010, pp. 505–516.
12. K. Chatterjee, T. Henzinger, and F. Horn, "Finitary Winning in omega-Regular Games." *TOCL*, vol. 11, no. 1, pp. 1:1–26, 2010.
13. K. Chatterjee, T. Henzinger, and M. Jurdziński, "Mean-Payoff Parity Games." in *LICS'05*. IEEE Computer Society, 2005, pp. 178–187.
14. K. Chatterjee, T. Henzinger, and N. Piterman, "Strategy Logic." *IC*, vol. 208, no. 6, pp. 677–693, 2010.
15. A. Condon, "The Complexity of Stochastic Games." *IC*, vol. 96, no. 2, pp. 203–224, 1992.
16. A. Ehrenfeucht and J. Mycielski, "Positional Strategies for Mean Payoff Games." *IJGT*, vol. 8, no. 2, 1979.

17. E. Emerson and C. Jutla, "Tree Automata, muCalculus, and Determinacy." in *FOCS'91*. IEEE Computer Society, 1991, pp. 368–377.
18. E. Emerson, C. Jutla, and A. Sistla, "On Model Checking for the muCalculus and its Fragments." in *CAV'93*, ser. LNCS 697. Springer, 1993, pp. 385–396.
19. ——, "On Model Checking for the muCalculus and its Fragments." *TCS*, vol. 258, no. 1-2, pp. 491–522, 2001.
20. E. Emerson and C.-L. Lei, "Temporal Reasoning Under Generalized Fairness Constraints." in *STACS'86*, ser. LNCS 210. Springer, 1986, pp. 267–278.
21. J. Fearnley, "Non-Oblivious Strategy Improvement." in *LPAR'10*, ser. LNCS 6355. Springer, 2010, pp. 212–230.
22. J. Fearnley and O. Lachish, "Parity Games on Graphs with Medium Tree-Width." in *MFCS'11*, ser. LNCS 6907. Springer, 2011, pp. 303–314.
23. J. Fearnley and S. Schewe, "Time and Parallelizability Results for Parity Games with Bounded Treewidth." in *ICALP'12*, ser. LNCS 7392. Springer, 2012, pp. 189–200.
24. M. Fellows and N. Koblitz, "Self-Witnessing Polynomial-Time Complexity and Prime Factorization." in *CSCT'92*. IEEE Computer Society, 1992, pp. 107–110.
25. ——, "Self-Witnessing Polynomial-Time Complexity and Prime Factorization." *DCC*, vol. 2, no. 3, pp. 231–235, 1992.
26. N. Fijalkow and M. Zimmermann, "Cost-Parity and Cost-Streett Games." in *FSTTCS'12*, ser. LIPIcs 18. Leibniz-Zentrum fuer Informatik, 2012, pp. 124–135.
27. ——, "Cost-Parity and Cost-Streett Games." *LMCS*, vol. 10, no. 2, pp. 1–29, 2014.
28. O. Friedmann and M. Lange, "Solving Parity Games in Practice." in *ATVA'09*, ser. LNCS 5799. Springer, 2009, pp. 182–196.
29. E. Grädel, W. Thomas, and T. Wilke, *Automata, Logics, and Infinite Games: A Guide to Current Research.*, ser. LNCS 2500. Springer, 2002.
30. V. Gurvich, A. Karzanov, and L. Khachivan, "Cyclic Games and an Algorithm to Find Minimax Cycle Means in Directed Graphs." *USSRCMMP*, vol. 28, no. 5, pp. 85–91, 1990.
31. M. Jurdziński, "Deciding the Winner in Parity Games is in UP ∩ co-Up." *IPL*, vol. 68, no. 3, pp. 119–124, 1998.
32. ——, "Small Progress Measures for Solving Parity Games." in *STACS'00*, ser. LNCS 1770. Springer, 2000, pp. 290–301.
33. M. Jurdziński, M. Paterson, and U. Zwick, "A Deterministic Subexponential Algorithm for Solving Parity Games." in *SODA'06*. Society for Industrial and Applied Mathematics, 2006, pp. 117–123.
34. ——, "A Deterministic Subexponential Algorithm for Solving Parity Games." *SJM*, vol. 38, no. 4, pp. 1519–1532, 2008.
35. N. Klarlund and D. Kozen, "Rabin Measures and Their Applications to Fairness and Automata Theory," in *LICS'91*. IEEE Computer Society, 1991, pp. 256–265.
36. O. Kupferman and M. Vardi, "Weak Alternating Automata and Tree Automata Emptiness." in *STOC'98*. Association for Computing Machinery, 1998, pp. 224–233.
37. A. Martin, "Borel Determinacy." *AM*, vol. 102, no. 2, pp. 363–371, 1975.
38. ——, "A Purely Inductive Proof of Borel Determinacy." in *SPM'82*, ser. Recursion Theory. American Mathematical Society and Association for Symbolic Logic, 1985, pp. 303–308.
39. R. McNaughton, "Infinite Games Played on Finite Graphs." *APAL*, vol. 65, pp. 149–184, 1993.
40. F. Mogavero, A. Murano, G. Perelli, and M. Vardi, "What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic." in *CONCUR'12*, ser. LNCS 7454. Springer, 2012, pp. 193–208.

41. ——, "Reasoning About Strategies: On the Model-Checking Problem." *TOCL*, vol. 15, no. 4, pp. 34:1–42, 2014.
42. F. Mogavero, A. Murano, and L. Sorrentino, "On Promptness in Parity Games." in *LPAR'13*, ser. LNCS 8312. Springer, 2013, pp. 601–618.
43. F. Mogavero, A. Murano, and M. Vardi, "Reasoning About Strategies." in *FSTTCS'10*, ser. LIPIcs 8. Leibniz-Zentrum fuer Informatik, 2010, pp. 133–144.
44. A. Mostowski, "Regular Expressions for Infinite Trees and a Standard Form of Automata." in *SCT'84*, ser. LNCS 208. Springer, 1984, pp. 157–168.
45. ——, "Games with Forbidden Positions." University of Gdańsk, Gdańsk, Poland, Tech. Rep., 1991.
46. J. Obdrzálek, "Fast Mu-Calculus Model Checking when Tree-Width Is Bounded." in *CAV'03*, ser. LNCS 2725. Springer, 2003, pp. 80–92.
47. ——, "Clique-Width and Parity Games." in *CSL'07*, ser. LNCS 4646. Springer, 2007, pp. 54–68.
48. S. Schewe, "Solving Parity Games in Big Steps." in *FSTTCS'07*, ser. LNCS 4855. Springer, 2007, pp. 449–460.
49. ——, "An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games." in *CSL'08*, ser. LNCS 5213. Springer, 2008, pp. 369–384.
50. ——, "ATL* Satisfiability is 2ExpTime-Complete." in *ICALP'08*, ser. LNCS 5126. Springer, 2008, pp. 373–385.
51. S. Schewe and B. Finkbeiner, "Satisfiability and Finite Model Property for the Alternating-Time muCalculus." in *CSL'06*, ser. LNCS 6247. Springer, 2006, pp. 591–605.
52. J. Vöge and M. Jurdziński, "A Discrete Strategy Improvement Algorithm for Solving Parity Games." in *CAV'00*, ser. LNCS 1855. Springer, 2000, pp. 202–215.
53. T. Wilke, "Alternating Tree Automata, Parity Games, and Modal muCalculus." *BBMS*, vol. 8, no. 2, pp. 359–391, 2001.
54. W. Zielonka, "Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees." *TCS*, vol. 200, no. 1-2, pp. 135–183, 1998.
55. U. Zwick and M. Paterson, "The Complexity of Mean Payoff Games on Graphs." *TCS*, vol. 158, no. 1-2, pp. 343–359, 1996.