Form Methods Syst Des (2018) 52:193–226
https://doi.org/10.1007/s10703-018-0315-1

Solving parity games via priority promotion

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

Published online: 31 January 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract We consider *parity games*, a special form of two-player infinite-duration games on numerically labeled 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 of identifying the corresponding winning regions 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 this decision problem, based on the idea of promoting vertexes 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.

Keywords Parity games · Infinite-duration games on graphs · Algorithmic complexity · Formal methods

1 Introduction

Parity games [49] are perfect-information two-player turn-based games of infinite duration, usually played on finite directed graphs. Their vertices, labeled by natural numbers called *priorities*, are called positions and assigned to one of two players, named *Even* and *Odd* or,

This work is based on [4], which appeared in CAV'16.

✉ Fabio Mogavero
fm@fabiomogavero.com

1 Università degli Studi di Napoli Federico I, Naples, Italy

2 University of Oxford, Oxford, UK

simply, 0 and 1, respectively. The game starts at an arbitrary position 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 position. The moves selected by the players induce an infinite sequence of positions, called play. If the maximal priority of the positions 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 [20,21], since these games are *memoryless determined* [19,41,42,49]. It has been even proved to belong to UPTIME ∩ COUPTIME [34], a status shared with the *factorization problem* [1,27,28]. They are the simplest class of games in a wider family with similar complexities and containing, e.g., *mean payoff games* [18,33], *discounted payoff games* [59], and *simple stochastic games* [17]. In fact, polynomial time reductions 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 et al. [20,21] have shown that computing winning strategies for these games is linear-time equivalent to solving the modal μCALCULUS model checking problem [22]. Parity games also play a crucial role in automata theory [19,40,48], where, for instance, they can be applied to solve the complementation problem for alternating automata [32] and the emptiness of the corresponding nondeterministic tree automata [40]. These automata, in turn, can be used to solve the satisfiability and model checking problems for expressive logics, such as the modal [57] and alternating [2,55] μCALCULUS, ATL* [2,54], Strategy Logic [16,44,45,47], Substructure Temporal Logic [5,6], and fixed-point extensions of guarded first-order logics [8,9].

Previous solutions mainly divide into two families: those that solve a game by first decomposing it into smaller subgames, and those that proceed in a global fashion and approach the game in its entirety. To the first family belongs the *divide et impera* solution originally proposed by McNaughton [43] for Muller games and adapted to parity games by Zielonka [58]. More recent improvements to that recursive algorithm have been proposed by Jurdziński et al. [37,38] and by Schewe [52]. 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 [35], which exploits the connection between the notions of progress measures [39] and winning strategies. In this approach, an initial measure function on the entire game is iteratively updated until a fixpoint is reached. At that point, a progress measure is obtained that induces a winning strategy for one of the two players. An alternative approach was proposed by Jurdziński and Vöge [56], which directly builds a winning strategy for one of the two players, by iteratively improving an initial non-winning strategy. This technique was later optimized by Schewe [53]. A recent breakthrough [11] by Calude *et al.* proposes a succinct reduction from parity to reachability games based on a clever encoding of the sequences of priorities a player finds along a play. This allows for a mere quasi-polynomial blow up in the size of the underlying graph and sets the basis of the fixed-parameter tractability w.r.t. the number of priorities. The approach has been then considerably refined in [24], where these encodings are modeled as progress measures. A similar technique is also used in [36]. Despite the theoretical relevance of this new idea, preliminary experiments conducted in this paper suggest that the practical impact of the result may not match the theoretical one. Indeed, most of

the exponential algorithms mentioned above outperform, often by orders of magnitude, the current implementations of the quasi-polynomial ones, which do not scale beyond a few hundred positions. This evaluation is consistent with the fact that the new techniques essentially amount to clever and succinct encodings embedded within a brute force search, which makes matching quasi-polynomial worst cases quite easy to find. As far as space consumption is concerned, we have different and, in some cases, incomparable behaviors. The small progress measure procedure of [35] requires $O(k \cdot n \cdot \log n)$ space, with n the number of positions in the game and k the number of its priorities. On the other hand, it is $O(n^2)$ for the optimized strategy improvement method of [53]. Due to their inherent recursive nature, the algorithms of the first family require $O(m \cdot n)$ memory, where m denotes the number of edges of the underlying graph. This bound could, in principle, be reduced to $O(n^2)$, by representing subgames implicitly through their sets of positions. The lowest space requirements, however, are those of the more recent quasi-polynomial algorithm described in [36], which only requires $O(n \cdot \log n \cdot \log k)$ additional space. 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 tree-width [25,26,50], dag-width [7], clique-width [51] and entanglement [10] 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 positions 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 move from a position 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–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, we provide an upper bound $O\left(kn\left(\frac{en}{k-1}\right)^{k-1}\right)$ on the time complexity, where e is Euler's number, and a bound $O(n \cdot \log k)$ on the memory requirements. Experimental results, comparing our algorithm with the state of the art solvers, also show that the proposed approach performs very well in practice, most often significantly better than existing ones, on both random games and benchmark families proposed in the literature.

2 Preliminaries

Let us briefly recall the notation and basic definitions concerning parity games that an expert reader can simply skip. We refer to [3,58] for a comprehensive presentation of the subject.

Given a partial function $\mathsf{f} : A \rightharpoonup B$, by $\mathsf{dom}(\mathsf{f}) \subseteq A$ and $\mathsf{rng}(\mathsf{f}) \subseteq B$ we indicate the domain and range of f, respectively. In addition, $\uplus$ denotes the *completion operator* that, taken f and another partial function $\mathsf{g} : A \rightharpoonup B$, returns the partial function $\mathsf{f} \uplus \mathsf{g} \triangleq (\mathsf{f} \setminus \mathsf{dom}(\mathsf{g})) \cup \mathsf{g} :$

$A \rightharpoonup B$, which is equal to g on its domain and assumes the same values of f on the remaining part of A.

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 without sinks. 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 $\mathsf{play}((\sigma^0, \sigma^1), v)$.

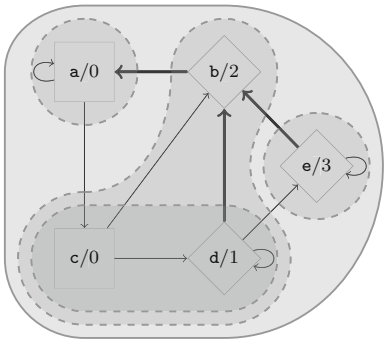
A *parity game* is a tuple $\mathcal{G} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathcal{P}$, where $\mathcal{A}$ is an arena, $\mathrm{Pr} \subset \mathbb{N}$ is a finite set of priorities, and $\mathsf{pr} : \mathrm{Ps} \to \mathrm{Pr}$ is a *priority function* assigning a priority to each position. The priority function can be naturally extended to games and paths as follows: $\mathsf{pr}(\mathcal{G}) \triangleq \mathsf{max}_{v \in \mathrm{Ps}}\, \mathsf{pr}(v)$; 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 $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ 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 $\mathcal{G} \setminus \mathrm{V}$ we denote the maximal subgame of $\mathcal{G}$ with set of positions Ps' contained in $\mathrm{Ps} \setminus \mathrm{V}$ 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, and 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 the set. For such a V, the set of positions of the subgame $\mathcal{G} \setminus \mathrm{V}$ is precisely $\mathrm{Ps} \setminus \mathrm{V}$. Finally, the set $\mathsf{esc}^\alpha(\mathrm{V}) \triangleq \mathsf{pre}^\alpha(\mathrm{Ps} \setminus \mathrm{V}) \cap \mathrm{V}$, called the α-*escape* of V, contains the positions in V from which α can leave V in one move. 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, instead, the α-positions from which α cannot escape with a single move. Observe that all the operators and sets described above actually depend on the specific game $\mathcal{G}$ they are applied in. In the rest of the paper, we shall only add $\mathcal{G}$ as subscript of an operator, e.g., $\mathsf{esc}^\alpha_{\mathcal{G}}(\mathrm{V})$, when the game is not clear from the context.

3 A new idea

A solution for a parity game $\mathcal{G} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathcal{P}$ over an arena $\mathcal{A} = \langle \mathrm{Ps}^0, \mathrm{Ps}^1, Mv \rangle$ 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 $\mathsf{atr}^\alpha_{\mathcal{G}}(\mathrm{D})$ gives an α-maximal dominion containing D.

Fig. 1 A simple game $\mathcal{G}$



In other words, α cannot force any position outside D to enter this set. The subgame $\mathcal{G}\setminus\mathsf{atr}^{\alpha}_{\mathcal{G}}(\mathrm{D})$ can then be solved by iterating the process. This procedure is reported in Algorithm 1. The crucial problem to address, therefore, consists in computing a dominion for some player in the game. The difficulty here is that, in general, no unique priority exists that satisfies the winning condition for a player along all the plays inside the dominion we are looking for. 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 passing through a specific set of positions, i.e., the escapes of the set itself. This notion is formalized by the following definition.

Algorithm 1: Parity game solver

```
signature sol_Γ : P → 𝒢 (2^{Ps_𝒢} × 2^{Ps_𝒢})
function sol_Γ(𝒢)
1   if Ps_𝒢 = ∅ then
2       return (∅, ∅)
    else
3       (R, α) ← src_Γ(𝒢)
4       R* ← atr^α_𝒢(R)
5       (W0', W1') ← sol_Γ(𝒢 \ R*)
6       (Wα, Wᾱ) ← (Wα' ∪ R*, Wᾱ')
7       return (W0, W1)
```

Definition 1 (*Quasi Dominion*) Let $\mathcal{G} \in \mathcal{P}$ be a game and $\alpha \in \{0, 1\}$ a player. A non-empty set of positions $\mathrm{Q} \subseteq \mathrm{Ps}_{\mathcal{G}}$ is a *quasi α-dominion* in $\mathcal{G}$ if there exists an α-strategy $\sigma_\alpha \in \mathrm{Str}^{\alpha}_{\mathcal{G}}(\mathrm{Q})$ such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}_{\mathcal{G}}(\mathrm{Q})$, with $\mathsf{int}^{\overline{\alpha}}_{\mathcal{G}}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\alpha}})$, and positions $v \in \mathrm{Q}$, the induced play $\pi = \mathsf{play}_{\mathcal{G}}((\sigma_0, \sigma_1), v)$ satisfies $\mathsf{pr}_{\mathcal{G}}(\pi) \equiv_2 \alpha$, if π is infinite, and $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\alpha}}_{\mathcal{G}}(\mathrm{Q})$, otherwise.

It is important to observe that the additional requirement that the opponent strategies be defined on all interior positions, formally $\mathsf{int}^{\overline{\alpha}}_{\mathcal{G}}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\alpha}})$, 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 $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Q}) \neq \emptyset$ (*resp.*, $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Q}) = \emptyset$). 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 $\mathcal{D}$. The set of pairs $(\mathrm{Q}, \alpha) \in 2^{\mathrm{Ps}_{\mathcal{D}}} \times \{0, 1\}$, where Q is a quasi α-dominion, is denoted by $\mathrm{QD}_{\mathcal{D}}$, and is partitioned into the sets $\mathrm{QD}^{-}_{\mathcal{D}}$ and $\mathrm{QD}^{+}_{\mathcal{D}}$ of open and closed quasi α-dominion pairs, respectively. As an example, consider Fig. 1. It is easy to see that the sets of positions $\{a\}$ and $\{b, c, d\}$ form two open quasi 0-dominion in the game $\mathcal{D}$. Similarly, the sets of positions $\{c, d\}$ and $\{e\}$ are open quasi 1-dominion. Finally, the entire game $\mathcal{D}$ is a closed quasi 0-dominion, or simply a 0-dominion, where the 0-strategy corresponds to the bold arrows.

An expert reader might note that quasi α-dominions are loosely related with the concept of *snares*, introduced in [23] and used there for completely different purposes, namely to speed up the convergence of standard 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 initialized 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. The pairs in such a relation also form 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 $\mathcal{D} \in \mathcal{P}$ is a tuple $\mathcal{D} \triangleq \langle \mathcal{D}, \mathcal{S}, \succ, \Re, \downarrow \rangle$, 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*) $\succ \subseteq \mathcal{S} \times \mathrm{QD}^{-}_{\mathcal{D}}$ is the *compatibility relation*, (*3*) $\Re : \mathrm{S} \to \mathrm{QD}_{\mathcal{D}}$ is the *query function* mapping each element $s \in \mathrm{S}$ to a quasi dominion pair $(\mathrm{Q}, \alpha) \triangleq \Re(s) \in \mathrm{QD}_{\mathcal{D}}$ such that, if $(\mathrm{Q}, \alpha) \in \mathrm{QD}^{-}_{\mathcal{D}}$ 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 bounded by the depth.

Different dominion spaces can be associated with the same game. Therefore, in the rest of this section, we shall simply assume a function Γ mapping every game $\mathcal{D}$ to a dominion space $\Gamma(\mathcal{D})$. Given the top element of $\mathcal{D} = \Gamma(\mathcal{D})$, Algorithm 2 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 $\mathcal{D}$, 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 $\mathsf{d}_{\mathcal{D}}(n, m, k)$ of the dominion space $\mathcal{D}$, where n, m, and k are the number of positions, moves, and priorities, respectively. Hence, $\mathsf{src}_{\mathcal{D}}$ runs in time $\mathrm{O}\big(\mathsf{d}_{\mathcal{D}}(n, m, k) \cdot (\mathsf{T}_{\Re}(n, m) + \mathsf{T}_{\downarrow}(n, m))\big)$, where $\mathsf{T}_{\Re}(n, m)$ and $\mathsf{T}_{\downarrow}(n, m)$ denote the time needed by the query and successor functions, respectively. Thus, the total time to solve

a game is $\mathrm{O}\big(m + n \cdot \mathsf{d}_{\mathcal{D}}(n, m, k) \cdot (\mathsf{T}_{\Re}(n, m) + \mathsf{T}_{\downarrow}(n, m))\big)$. 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 m, the whole procedure terminates in time $\mathrm{O}(n \cdot (n + m) \cdot \mathsf{d}_{\mathcal{D}}(n, m, k))$. 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 $\mathcal{D}$.

Algorithm 2: The searcher

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

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.

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 $\mathsf{pr}(\partial) \equiv_2 \alpha$ and all the positions in $\mathsf{esc}^{\overline{\alpha}}_{\partial}(\mathrm{R})$ have priority $\mathsf{pr}(\partial)$, i.e. $\mathsf{esc}^{\overline{\alpha}}_{\partial}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\partial}(\mathsf{pr}(\partial))$.

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 Rg_{∂} the set of region pairs in ∂ and with $\mathrm{Rg}^{-}_{\partial}$ and $\mathrm{Rg}^{+}_{\partial}$ the sets of open and closed region pairs, respectively. A closed α-region is clearly an α-dominion. As an example, consider again Fig. 1. The singleton $\{e\}$ is the unique 1-region in the entire game ∂, which is also open, while $\{b\}$ is a non-maximal 0-region in the subgame $\partial \setminus \{e\}$ where e is removed. Finally, the set $\{b, c, d\}$ is a maximal open 0-region in the same subgame.

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* $\mathcal{D} \in \mathcal{P}$ *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 $\mathsf{int}^{\overline{\alpha}}_{\mathcal{D}\setminus\mathrm{R}}(\mathrm{D}) \subseteq \mathsf{dom}\big(\sigma'_{\overline{\alpha}}\big)$, 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 $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(\mathsf{pr}(\mathcal{D}))$, which has the maximal priority in $\mathcal{D}$ 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 $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}) \setminus \mathsf{pre}^{\alpha}_{\mathcal{D}}(\mathrm{D}) = \mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$. Therefore, $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}^\star)$. 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 $\mathsf{atr}^{\alpha}_{\mathcal{D}}(\mathrm{R}^\star) \setminus \mathrm{R}^\star$, 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 $\mathsf{atr}^{\alpha}_{\mathcal{D}}(\mathrm{R}) \setminus \mathrm{R}$ or $\mathsf{atr}^{\alpha}_{\mathcal{D}\setminus\mathrm{R}}(\mathrm{D}) \setminus \mathrm{D}$, 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 $v \in \mathsf{atr}^{\alpha}_{\mathcal{D}}(\mathrm{R}) \setminus \mathrm{R}$, contradicting α-maximality of R in $\mathcal{D}$. If not, then in the subgame $\mathcal{D} \setminus \mathrm{R}$, the remaining moves from v must all lead to D. But then, $v \in \mathsf{atr}^{\alpha}_{\mathcal{D}\setminus\mathrm{R}}(\mathrm{D}) \setminus \mathrm{D}$, contradicting α-maximality of D in $\mathcal{D} \setminus \mathrm{R}$. □

During the search, we keep track of the computed regions by means of an auxiliary priority function $\mathsf{r} \in \mathbb{P}_{\mathcal{D}} \triangleq \mathrm{Ps}_{\mathcal{D}} \to \mathrm{Pr}_{\mathcal{D}}$, called *region function*, which formalizes the intuitive notion of priority of a region described above. Initially, the region function coincides with the priority function $\mathsf{pr}_{\mathcal{D}}$ of the entire game $\mathcal{D}$. 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 $\mathrm{R} \triangleq \mathsf{atr}^{\alpha}_{\mathcal{D}^{\star}}\big(\mathsf{r}^{-1}(p)\big)$ is computed w.r.t. the subgame $\mathcal{D}^{\star}$, which is derived from $\mathcal{D}$ 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 $\mathcal{D}^{\star}$, 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 $\mathcal{D}$ 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 $\mathcal{D}^{\star}$, 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 $\mathsf{pr}_{\mathcal{D}}$ 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 formalized by the following proposition.

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

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. □

Figure 2 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

Fig. 2 Running example

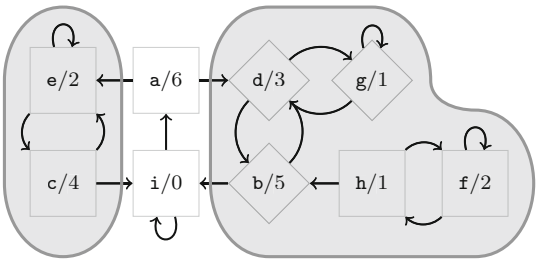


Table 1 PP simulation

	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$\uparrow_6$
3	d↓	d↓	d,g$\uparrow_5$				
2	e$\uparrow_4$			e$\uparrow_4$		e, f, h$\uparrow_4$	
1		g$\uparrow_3$					
0					i$\uparrow_6$		

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 {a}, which is open because of the two moves leading to d and e. At priority 5, the open 1-region {b, f, h} is formed by attracting both f and h to b, which is open in the subgame where {a} is removed. Similarly, the 0-region {c} at priority 4 and the 1-region {d} at priority 3 are open, once removed {a, b, f, h} and {a, b, c, f, h}, respectively, from the game. At priority 2, the 0-region {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 {e} to 4 is then performed, resulting in the new 0-region {c, 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 {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 {a, b, d, g, i} and the closed region 4 containing {c, e, f, 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 {c, e} are reset to their initial priorities, when 1-region {d, g} in Column 3 is promoted to 5. Similarly, when 0-region {i} in Column 5 is promoted to 6, the priorities of the positions in both regions {b, d, f, g, h} and {c, e}, highlighted by

the gray areas, are reset. This is actually necessary for correctness, at least in general. In fact, if region {b, d, f, g, h} were not reset, the promotion of {i} to 6, which also attracts b, d, and g, would leave {f, 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 formalize 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.

4.1 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 $\mathsf{r} \in \mathbb{P}_{\mathcal{D}}$ 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, $\mathsf{r}^{(\geq p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) \geq p\}$ (*resp.*, $\mathsf{r}^{(> p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) > p\}$ 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 $\mathsf{r} \in \mathbb{R}_{\mathcal{D}} \subseteq \mathbb{P}_{\mathcal{D}}$ 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. For instance, the state s corresponding to Row 4 Column 4 in Table 1 is $(\mathsf{r}, 4)$, where the region function r is such that $\mathsf{r}(\mathtt{a}) = 6$, $\mathsf{r}(x) = 5$, for $x \in \{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$, and $\mathsf{r}(x) = \mathsf{pr}(x)$, for $x \in \{\mathtt{c}, \mathtt{e}, \mathtt{i}\}$. In addition, the subgame $\mathcal{D}_s$ of s only contains the positions c, e, and i.

We can now formally define the *Priority Promotion* dominion space, by characterizing 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 $\mathcal{S}_{\mathcal{D}} \triangleq \langle \mathrm{S}_{\mathcal{D}}, \top_{\mathcal{D}}, \prec_{\mathcal{D}} \rangle$, where its components are defined as prescribed in the following:

1. $\mathrm{S}_{\mathcal{D}} \subseteq \mathbb{R}_{\mathcal{D}} \times \mathrm{Pr}_{\mathcal{D}}$ is the set of all pairs $s \triangleq (\mathsf{r}, p)$, called *states*, composed of a region function $\mathsf{r} \in \mathbb{R}_{\mathcal{D}}$ and a priority $p \in \mathrm{Pr}_{\mathcal{D}}$ such that (*a*) r is maximal above p, (*b*) $p \in \mathsf{rng}(\mathsf{r})$, and (*c*) $\mathsf{r}^{(< p)} \subseteq \mathsf{pr}_{\mathcal{D}}{}^{(< p)}$;
2. $\top_{\mathcal{D}} \triangleq (\mathsf{pr}_{\mathcal{D}}, \mathsf{pr}(\mathcal{D}))$;
3. for any two states $s_1 \triangleq (\mathsf{r}_1, p_1), s_2 \triangleq (\mathsf{r}_2, p_2) \in \mathrm{S}_{\mathcal{D}}$, it holds that $s_1 \prec_{\mathcal{D}} s_2$ 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 $\mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\ni \leq q}$. 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 $\mathsf{pr}_{\ni}$ 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 q that strictly contains the corresponding one in s_2 and all regions above q are equal in the two states, or state s_1 is currently processing a lower priority than the one of s_2.

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 $\ni_s$. 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 $(\mathrm{R}, \alpha) \in \mathrm{QD}^{-}_{\ni}$ is *compatible* with a state $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_{\ni}$, in symbols $s \succ_{\ni} (\mathrm{R}, \alpha)$, iff (*1*) $(\mathrm{R}, \alpha) \in \mathrm{Rg}_{\ni_s}$ and (*2*) if R is α-open in $\ni_s$ then (*2.a*) R is α-maximal in $\ni_s$ and (*2.b*) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$.

Algorithm 3: 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)
```

Algorithm 3 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 $\ni_s$ 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 $\ni_s$ 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 $\ni_s$ and a region function r in the whole game $\ni$. 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 formalize this concept, let $I \triangleq Mv_{\ni} \cap ((\mathrm{R} \cap \mathrm{Ps}^{\overline{\alpha}}_{\ni}) \times (\mathsf{dom}(\mathsf{r}) \setminus \mathrm{R}))$ 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, $\mathsf{bep}^{\overline{\alpha}}_{\ni}(\mathrm{R}, \mathsf{r})$ is set to the minimal priority among those regions containing positions reachable by a move in I. Formally, $\mathsf{bep}^{\overline{\alpha}}_{\ni}(\mathrm{R}, \mathsf{r}) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r}{\restriction}\mathsf{rng}(I)))$. Note that, if R is a closed α-region in $\ni_s$, then $\mathsf{bep}^{\overline{\alpha}}_{\ni}(\mathrm{R}, \mathsf{r})$ 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 Fig. 2, for 0-region

$\mathrm{R} = \{\mathtt{e}, \mathtt{f}, \mathtt{h}\}$ with priority equal to 2 in column 6, we have that $I = \{(\mathtt{e}, \mathtt{c}), (\mathtt{h}, \mathtt{b})\}$ and $\mathsf{r}{\restriction}\mathsf{rng}(I) = \{(\mathtt{c}, 4), (\mathtt{b}, 6)\}$. Hence, $\mathsf{bep}^1_{\mathcal{D}}(\mathrm{R}, \mathsf{r}) = 4$.

In order to perform a reset of the priority of some positions in the game after a promotion, we use the completing operator $\uplus$. Taken two partial function $\mathsf{f}, \mathsf{g} \colon \mathrm{A} \rightharpoonup \mathrm{B}$, the operator returns the partial function $\mathsf{f} \uplus \mathsf{g} \colon \mathrm{A} \rightharpoonup \mathrm{B}$, which is equal to g on its domain and assumes the same values as g on the remaining part of the set A.

Algorithm 4: 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*)
```

Algorithm 4 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 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.

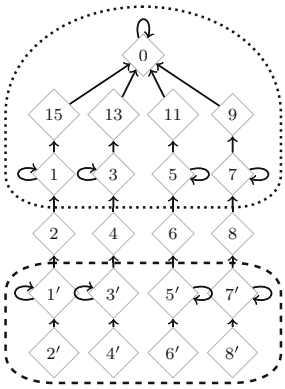
As already observed in Sect. 3, a dominion space, together with Algorithm 2, provides a sound and complete solution procedure. The following theorem states that the priority-promotion mechanism presented above is indeed a dominion space. For the sake of readability, the proof is provided in the "Appendix".

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 3 and 4 is a dominion space.*

4.2 Complexity of PP dominion space

To assess the complexity of the PP approach, we produce an estimate of the size and depth of the dominion space $\mathcal{R}_{\mathcal{D}}$. This provides upper bounds for both the time and space complexities needed by the search procedure $\mathsf{src}_{\mathcal{R}_{\mathcal{D}}}$ that computes 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+1}. 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. Moreover, every such function can be associated with at most k current priorities.

Fig. 3 The $\partial_{2,4}^{\mathrm{PP}}$ game



Measuring the depth is a little trickier. A relatively coarse bound can be obtained by observing that there is an homomorphism from $\mathcal{S}_\partial$ to the well-founded partial order, in which the region function r of a state is replaced by a partial function $\mathsf{f} : \mathrm{Pr}_\partial \rightharpoonup [1, n]$ 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 $\mathcal{S}_\partial$ 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, thanks to Stirling's approximation of the factorial function, every chain cannot be longer than $k\binom{n+k}{k} = (n+k)\binom{n+k-1}{k-1} \leq (n+k)\,(\mathrm{e}(n/(k-1)+1))^{k-1}$, where e is Euler's number.

Theorem 2 (Size and Depth Upper Bounds) *The size and depth of a* PP *dominion space* $\mathcal{R}$ *with* $n \in \mathbb{N}_+$ *positions and* $k \in [1, n]$ *priorities are bounded by* k^{n+1} *and* $(n+k)\,(\mathrm{e}(n/(k-1)+1))^{k-1} = \mathrm{O}\left(kn\left(\frac{\mathrm{e}n}{k-1}\right)^{k-1}\right)$, *respectively.*

Unfortunately, due to the reset operations performed after each promotion, an exponential worst-case can actually be built. Indeed, consider the game $\partial_{l,h}$ having all positions ruled by player 0 and containing h chains of length $2l+1$ that converge into a single position of priority 0 with a self loop. The i-th chain has a head of priority $2(2h-i)+1$ and a body composed of l blocks of two positions with priority $2i-1$ and $2i$, respectively. The first position in each block also has a self loop. An instance of this game with $l = 2$ and $h = 4$ is depicted in Fig. 3. The labels of the positions, deprived of the possible apexes, 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 l. 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 l. Hence, the overall number of performed promotions is $(l+1)^h$. 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 apexes 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 maximized 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 maximized 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 maximized 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 l. Thus, after an exponential number promotions, the closed 0-region $\{0\}$ is eventually obtained as solution.

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

Proof The single-player game $\mathfrak{I}_h^{\mathrm{PP}}$, with $k = 2h + 1$ positions and priorities, contains a position with priority 0 plus $2h$ positions spanning all the odd priorities from 1 to $4h - 1$. Moreover, the associated moves are those depicted in the highlighted top section of Fig. 3. Intuitively, this game encodes a binary counter, where each one of the h chains, composed of two positions of odd priorities connected to position 0, represents a digit. The promotion operation of region $2i-1$ to region $2(2h-i)+1$ corresponds to the increment of the i-th digit. As a consequence, the number of promotions for the PP algorithm is equal to the number of configurations of the counter, except for the initial one. Now, to provide a lower bound for the depth of the associated dominion space $\mathcal{D}_h^{\mathrm{PP}}$, consider the recursive function $\mathsf{Q}(h)$ with base case $\mathsf{Q}(0) = 1$ and inductive case $\mathsf{Q}(h) = 2\mathsf{Q}(h - 1) + 2$. This function counts the number of queries executed by the the PP algorithm on game $\mathfrak{I}_h^{\mathrm{PP}}$. The correctness of the base case is trivial. Indeed, when h is equal to 0, there are no chains and the only possible region is the one containing position 0. Let us now consider the inductive case $h > 0$. A first query is required to obtain region $\{4h-1\}$. Then, before reaching the promotion of region $\{1\}$ to region $\{4h-1\}$, i.e., to set the most-significant digit, all the other digits must be set. To do this, the number of necessary queries is equal to those required on the game instance with $h-1$ chains minus the query that computes region $\{0\}$, i.e., $\mathsf{Q}(h - 1) - 1$. At this point, two additional queries are counted, when region $\{1\}$ is obtained and then merged to region $\{4h - 1\}$. As described above in the execution of the game depicted in Fig. 3, this promotion unnecessarily resets all lesser-significant digits. Therefore, $\mathsf{Q}(h - 1) - 1$ counts exactly the queries needed to set again to 1 all the digits reset after this promotion. Finally, a query for the 0-closed region $\{0\}$ is required. Summing up, we have $\mathsf{Q}(h) = 1 + (\mathsf{Q}(h-1)-1) + 2 + (\mathsf{Q}(h-1)-1) + 1 = 2\mathsf{Q}(h-1)+2$, as claimed above. A trivial proof by induction shows that $\mathsf{Q}(h) = 3 \cdot 2^h - 2 = \Theta\big(2^{k/2}\big)$.

We can now turn to the more complex single-player game $\mathfrak{I}_{l,h}^{\mathrm{PP}}$, with $n = (2l + 1) \cdot h + 1$ positions and $k = 3h + 1$ priorities, which contains a position with priority 0 plus $(2l + 1) \cdot h$ positions, spanning all the priorities from 1 to $2h$ and all odd priorities from $2h + 1$ to $4h - 1$ as depicted in the entire Fig. 3. To provide a lower bound for the depth of the associated dominion space $\mathcal{D}_{l,h}^{\mathrm{PP}}$, consider the recursive function $\mathsf{Q}(h)$ with base case $\mathsf{Q}(l, 0) = 1$ and inductive case $\mathsf{Q}(l, h) = (1 + l)\mathsf{Q}(l, h - 1) + 2l + 1$. Also in this case, the correctness of the base cases is trivial. Indeed, when h is equal to 0, there are no chains, independently of the

parameter l. Hence, the only possible region is the one containing position 0. For the inductive case $h > 0$, a first query is required to compute region $\{4h-1\}$. Then, before reaching the promotion to region $\{4h-1\}$ of the adjacent region $\{1\}$, i.e., to set the most-significant digit to 1, all the other digits must be set to $l-1$. This requires the same number of queries required to process the game with $h-1$ chains minus the query collecting region $\{0\}$, i.e., $\mathsf{Q}(l, h-1)-1$. At this point, three additional queries are counted: region $\{2, 1', 2', \ldots\}$ is computed first, followed by region $\{1\}$; the latter is then merged to region $\{4h-1\}$ absorbing position 2 from the first region. Note that the region of priority 2 contains all positions in the first chain, except for positions 1 and $4h-1$. As in the previous game, such a promotion unnecessarily resets all lesser-significant digits. Therefore, $\mathsf{Q}(l, h-1)-1$ counts exactly the queries needed to set again to $l-1$ all digits reset after this promotion. After that, and similarly to what is described above, we need to set the most-significant digit to 2, by performing other three queries: $\{2', \ldots\}$ is computed first, followed by region $\{1'\}$; the latter is then merged to region $\{4h-1, 1, 2\}$, absorbing position $2'$ from the first region. Again, this promotion resets all lesser-significant digits, which requires $\mathsf{Q}(l, h-1)-1$ steps to be set to $l-1$. This behavior is repeated $l-2$ more times, until the first chain is completely absorbed by the region of highest priority $4h-1$. Finally, a query for the 0-closed region $\{0\}$ is required. Summing up, we have $\mathsf{Q}(n, h) = 1 + (\mathsf{Q}(h-1)-1) + (3 + (\mathsf{Q}(l, h-1)-1)) \cdot l + 1 = (1+l)\mathsf{Q}(l, h-1) + 2l + 1$, as claimed above. Also in this case, an easy induction shows that $\mathsf{Q}(l, h) = ((3l+1) \cdot (l+1)^h - 1)/l - 2 = \mathrm{O}\big((3n/(2(k-1)))^{k/3}\big)$. □

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 $\mathcal{D}_{l,h}$ described in the previous paragraph, where $k = 3h+1$. The other bound, instead, has a base fixed to 2, but a larger 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 behavior comparable with the one exhibited by the small-progress measure procedure [35].

5 Subgame decomposition

As described in the previous section, the search procedure for the basic PP approach, displayed in Algorithm 2, consists in finding quasi dominions that, whenever closed in the current subgame, are suitably merged with previously computed regions until a dominion in the original game $\mathcal{D}$ is found. In particular, when the current quasi dominion is open, the algorithm searches for the next one in a subgame $\mathcal{D}'$ of $\mathcal{D}$. By Algorithm 3 and Lines 2-3 of Algorithm 4, such a subgame corresponds to $\mathcal{D}_s \setminus \mathrm{R}$, where $\mathcal{D}_s$ and R are, respectively, the current subgame w.r.t. to state s and the region extracted by the query function on s itself. Due to this design choice and the fact that the regions are created in decreasing order of priority, the sequence of subgames follows the same order and, in particular, the current priority p contained in the state $s = (\mathsf{r}, p)$ also corresponds to the maximal priority in the part of game yet to be analyzed. This is a very simple and efficient way to compute subgames, but it does not necessarily translate into an efficient way to obtain a solution in terms of the number of promotions required. In fact, the lesser the moves between $\mathcal{D}'$ and the positions outside $\mathcal{D}'$, the easier it is to find a dominion in the subsequent iterations, as the probability that a promotion occurs reduces. In this section we propose a generalization of the PP algorithm that does not commit to a specific way of computing the next subgame to process during its dominion search phase. The result is a parametric version of the priority promotion approach that, by exploiting the

topology of the underlying game, can be instantiated with different game decomposition techniques.

To this end, we first need to address the crucial issue that the correctness of promotion mechanism relies, according to Proposition 1, on the fact that the positions contained in a dominion to be promoted can only reach some previously computer region. To ensure that this property is preserved, we need to require that, after computing region R in game $\mathcal{D}_s$, whatever the subgame $\mathcal{D}'$ of $\mathcal{D}_s \setminus \mathrm{R}$ we chose to consider next, it cannot contain positions that reach portions of the game yet to be analyzed, namely positions in the set $\mathrm{Ps}_{\mathcal{D}_s} \setminus (\mathrm{R} \cup \mathrm{Ps}_{\mathcal{D}'})$. Since there is no unique way to choose a suitable subgame, the new algorithm we propose in the following is parametric on a function F that, given a game $\mathcal{D}$, computes the positions of the subgame $\mathcal{D}'$ on which the search for quasi dominions needs to proceed. The standard PP algorithm can be obtained by simply setting $\mathsf{F}(\mathcal{D}) = \mathrm{Ps}_{\mathcal{D}}$. A sharper heuristic, instead, instantiates F with a procedure that collects the positions in some minimal Strongly Connected Component (SCC) of the graph underlying the game $\mathcal{D}$. An alternative coarser possibility, which, however, incurs in less overhead, simply computes all the positions reachable from an arbitrary initial position in the game. It is immediate to see that all these instantiations satisfy the above requirement on the subgames.

A second issue concerns the way in which the processed subgames are kept track of. In the original PP algorithm, where $\mathcal{D}' = \mathcal{D}_s \setminus \mathrm{R}$, the current subgame $\mathcal{D}_s$ coincides with $\mathcal{D}_{\mathsf{r}}^{\leq p}$. Hence, the region function r, together with the priority p, suffices to identify the current subgame $\mathcal{D}_s$. On the other hand, if $\mathcal{D}'$ is chosen as a strict subgame of $\mathcal{D}_{\mathsf{r}}^{\leq p}$, we need to introduce an auxiliary function to keep track of the computed subgames and identify the current one. This role is played by a second priority function $\mathsf{g} : \mathrm{Ps} \to \mathrm{Pr}$, called *subgame function*. At the beginning of the procedure, g maps every position to $\mathsf{pr}(\mathcal{D})$ in order to ensure that the current subgame is the entire game. Every time a new subgame $\mathcal{D}'$ of $\mathcal{D}_s$ is computed, the subgame function g is updated by setting all positions contained in $\mathcal{D}'$ to the maximal priority p' of $\mathcal{D}'$ itself. Essentially, g induces a sequence of subgames ordered by game inclusion and indexed by the maximal priorities in each subgame. Consequently, g maps each position to the priority identifying the minimal subgame in the sequence that contains that position. As a result, we have that $\mathrm{Ps}_{\mathcal{D}'} = \mathsf{g}^{-1}(p')$ and $\mathrm{Ps}_{\mathcal{D}_s} = \mathsf{dom}\big(\mathsf{g}^{(\leq p)}\big)$. In this new setting, the current priority p of each state can be recovered as the minimal priority in the range of g, while the current subgame corresponds to $\mathcal{D}_{\mathsf{g}}^{\leq p} \triangleq \mathcal{D} \restriction \mathsf{dom}\big(\mathsf{g}^{(\leq p)}\big)$. Formally, a priority function $\mathsf{g} \in \mathbb{G}_{\mathcal{D}} \subseteq \mathrm{Ps}_{\mathcal{D}} \to \mathrm{Pr}_{\mathcal{D}}$ is a *subgame function* iff (*i*) $\mathsf{rng}(\mathsf{g}) \subseteq \mathsf{rng}\big(\mathsf{pr}_{\mathcal{D}}\big)$ and (*ii*), for all $p \in \mathsf{rng}(\mathsf{g})$, it holds that (*ii.a*) $\mathcal{D}_{\mathsf{g}}^{\leq p} \triangleq \mathcal{D} \restriction \mathsf{dom}\big(\mathsf{g}^{(\leq p)}\big)$ is a game and (*ii.b*) $\mathsf{g}^{(\leq p)} \subset \mathsf{g}^{(\leq p')}$, for all $p' \in \mathsf{rng}(\mathsf{g})$ with $p < p'$.

A final issue involves the way we keep track of the computed regions. In the PP procedure, we use region function r, which is initialized to $\mathsf{pr}_{\mathcal{D}}$ and directly stores the stratification of the regions computed in decreasing order of priority. Therefore, for every priority p, the set $\mathsf{r}^{-1}(p)$ exactly corresponds to a region R in $\mathcal{D}_s = \mathcal{D}_{\mathsf{r}}^{\leq p}$, where $s \triangleq (\mathsf{r}, p)$. If the subgame $\mathcal{D}_s$ is computed via the subgame function g, instead, a problem with the original definition of the region function r arises: r might contain positions associated with priorities that may not exist in g. Note that this problem is a direct consequence of the strict inclusion $\mathcal{D}' \subset \mathcal{D}_s \setminus \mathrm{R}$. For this reason, we generalize the notion of region function r to a *quasi-dominion function* q. The regions computed so far can still be recovered by restricting q to the current subgame, identified by g, in symbols $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. Formally, given a game $\mathcal{D}$, a priority function $\mathsf{q} \in \mathbb{Q}_{\mathcal{D}} \subseteq \mathrm{Ps}_{\mathcal{D}} \to \mathrm{Pr}_{\mathcal{D}}$ is a *quasi-dominion function* iff, for all $p \in \mathsf{rng}(\mathsf{q})$, it holds that $\mathsf{q}^{-1}(p)$ is an α-quasi dominion in $\mathcal{D}$, with $\alpha \triangleq p \bmod 2$. We say that the function $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$ is a g-*stratified region function* if (*i*) $\mathsf{r}^{-1}(p)$ is an α-region in $\mathcal{D}_{\mathsf{g}}^{\leq p}$, for all

$p \in \mathsf{rng}(\mathsf{r})$, and (*ii*) $\mathsf{esc}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\alpha}}_{\mathfrak{D}}(\mathrm{X}) = \emptyset$, for each $p \in \mathsf{rng}(\mathsf{g})$ and quasi dominion pair $(\mathrm{Q}, \alpha) \in \mathrm{QD}_{\mathfrak{D}^{\leq p}_{\mathsf{g}}}$, where $\mathrm{X} = \mathrm{Ps}_{\mathfrak{D}} \setminus (\mathsf{dom}(\mathsf{g}^{(\leq p)}) \cup \mathsf{dom}(\mathsf{r}))$. Moreover, r is *maximal* w.r.t. g if $\mathsf{r}^{-1}(p)$ is α-maximal in $\mathfrak{D}^{\leq p}_{\mathsf{g}}$, for all $p \in \mathsf{rng}(\mathsf{r})$ with $p > \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ and $\alpha = p \bmod 2$. Intuitively, Items (*i*) of the g-stratified property and the maximality of r w.r.t. g state the same two requirements that a classic PP region function must satisfy. In addition, Item (*ii*) of the former property precisely formalizes the above described requirement on the quasi dominions: every quasi dominion in a subgame identified by the function g cannot have positions with moves that lead outside the domain of the function r.

In the following, we adapt the PP dominion space to support the new subgame decomposition approach. A state is now defined as a pair $s = (\mathsf{q}, \mathsf{g})$, where q is a quasi dominion function and g is a subgame function. In addition, the game induced by s is defined as $\mathfrak{D}_s \triangleq \mathfrak{D}^{\leq p}_{\mathsf{g}}$, with $p = \mathsf{min}(\mathsf{rng}(\mathsf{g}))$.

Definition 6 (*State Space*) A *state space* is a tuple $\mathcal{S}_{\mathfrak{D}} \triangleq \langle \mathrm{S}_{\mathfrak{D}}, \top_{\mathfrak{D}}, \prec_{\mathfrak{D}} \rangle$, where its components are defined as prescribed in the following:

1. $\mathrm{S}_{\mathfrak{D}} \subseteq \mathbb{Q}_{\mathfrak{D}} \times \mathbb{G}_{\mathfrak{D}}$ is the set of all pairs $s \triangleq (\mathsf{q}, \mathsf{g})$, called *states*, composed of a quasi dominion function $\mathsf{q} \in \mathbb{Q}_{\mathfrak{D}}$ and a subgame function $\mathsf{g} \in \mathbb{G}_{\mathfrak{D}}$ such that (*a*) $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$ is maximal w.r.t. g, (*b*) r is a g-stratified region function, and (*c*) $\mathsf{q} = \mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{r}$;
2. $\top_{\mathfrak{D}} \triangleq (\mathsf{pr}_{\mathfrak{D}}, \varnothing[\mathrm{Ps}_{\mathfrak{D}} \mapsto \mathsf{pr}(\mathfrak{D})])$;
3. for any two states $s_1 \triangleq (\mathsf{q}_1, \mathsf{g}_1)$, $s_2 \triangleq (\mathsf{q}_2, \mathsf{g}_2) \in \mathrm{S}_{\mathfrak{D}}$, with $p_1 \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}_1))$ and $p_2 \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}_2))$, it holds that $s_1 \prec_{\mathfrak{D}} s_2$ iff either (*a*) there exists a priority $p \in \mathsf{rng}(\mathsf{q}_1)$ with $p \geq p_1$ such that (*a.i*) $\mathsf{q}_1{}^{(>p)} = \mathsf{q}_2{}^{(>p)}$ and (*a.ii*) $\mathsf{q}_2^{-1}(p) \subset \mathsf{q}_1^{-1}(p)$ or (*b*) both (*b.i*) $\mathsf{q}_1 = \mathsf{q}_2$ and (*b.ii*) $p_1 < p_2$ hold.

As one would expect, the state space defined above is slightly different from the one in Sect. 4. Indeed, a state (q, g) does not explicitly record the regions anymore. Instead, it records the quasi dominions from which the computed regions can be extracted. In more detail, an α-maximal region is obtained from the derived set $\mathsf{r}^{-1}(q) = \mathsf{q}^{-1}(q) \cap \mathsf{g}^{-1}(q)$, where $q \geq \mathsf{min}(\mathsf{rng}(\mathsf{g}))$, $\alpha = q \bmod 2$, and $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. Obviously, the current priority $p \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ of the state must be a priority of an actual region in r. Moreover, the quasi dominion function q, at any priority p' lower than p, must contain positions with the same priority p' in the original priority function $\mathsf{pr}_{\mathfrak{D}}$, i.e., $\mathsf{q}^{-1}(p') \subseteq \mathsf{pr}_{\mathfrak{D}}^{-1}(p')$. At any p' grater that p, instead, q needs to store part of the priority function together with the regions, that is $\mathsf{q}^{-1}(p') = (\mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{r})^{-1}(p')$. All these requirement are ensured by Points (*a*)–(*c*) of Item 1. Finally, the order follows exactly the same rules as in the original algorithm, where the pair $(\mathsf{q} \cap \mathsf{g}, \mathsf{min}(\mathsf{rng}(\mathsf{g})))$ plays the role of the PP state (r, p).

The compatibility relation is very similar to the one for the PP procedure. Indeed, we only need to specify how to extract the function r from the state s.

Definition 7 (*Compatibility Relation*) An open quasi dominion pair $(\mathrm{R}, \alpha) \in \mathrm{QD}^{-}_{\mathfrak{D}}$ is *compatible* with a state $s \triangleq (\mathsf{q}, \mathsf{g}) \in \mathrm{S}_{\mathfrak{D}}$, in symbols $s \succ_{\mathfrak{D}} (\mathrm{R}, \alpha)$, iff (*1*) $(\mathrm{R}, \alpha) \in \mathrm{Rg}_{\mathfrak{D}_s}$ and (*2*) if R is α-open in $\mathfrak{D}_s$ then (*2.a*) R is α-maximal in $\mathfrak{D}_s$ and (*2.b*) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$, where $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$.

Algorithm 5: New query function

signature $\Re_{\mathcal{D}} : \mathrm{S}_{\mathcal{D}} \to (2^{\mathrm{Ps}_{\mathcal{D}}} \times \{0, 1\})$
function $\Re_{\mathcal{D}}(s)$
  let $(\mathsf{q}, \mathsf{g}) = s$ **in**
1 $p \leftarrow \mathsf{min}(\mathsf{rng}(\mathsf{g}))$
2 $\alpha \leftarrow p \bmod 2$
3 $\mathrm{R} \leftarrow \mathsf{atr}^{\alpha}_{\mathcal{D}_s}\left(\mathsf{q}^{-1}(p) \cap \mathrm{Ps}_{\mathcal{D}_s}\right)$
4 **return** (R, α)

Algorithm 5 provides the implementation for the query function compatible with the generalized priority-promotion mechanism. Let $s \triangleq (\mathsf{q}, \mathsf{g})$ be the current state. Line 1 extracts from s the priority p to process, while Line 2 simply computes the associated parity α. Finally, Line 3 computes the attractor w.r.t. player α of the quasi dominion contained in q at p that belongs to the subgame $\mathcal{D}_s$, i.e., the α-region $\mathsf{q}^{-1}(p) \cap \mathrm{Ps}_{\mathcal{D}_s} = \mathsf{q}^{-1}(p) \cap \mathsf{g}^{-1}(p) = \mathsf{r}^{-1}(p)$. The resulting set R is, according to Proposition 2, an α-maximal α-region in $\mathcal{D}_s$ that contains $\mathsf{r}^{-1}(p)$.

Unlike for the classic PP algorithm, the notion of *best escape priority* for player $\overline{\alpha}$ w.r.t. an α-region R of the subgame $\mathcal{D}_s$ is defined w.r.t. the quasi dominion function q. Indeed, we do not need to consider the g-stratified region function r, since, due to the property of the subgame function g, the escape positions in every quasi dominion in $\mathcal{D}_s$ can only reach positions in r. This means that the range of the interface relation I is included in r itself. At this point, the successor function can be defined parametrically on the function F that identifies a subgame $\mathcal{D}' \subseteq \mathcal{D} \setminus \mathrm{R}$ such that, for each player α and quasi α-dominion $\mathrm{Q} \in \mathrm{QD}_{\mathcal{D}'}$, the set $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}'}(\mathrm{Q})$ has no positions that reach $\mathcal{D} \setminus (\mathrm{R} \cup \mathrm{Ps}_{\mathcal{D}'})$. More formally, the set $\mathrm{A} \triangleq \mathsf{F}(\mathcal{D}) \subseteq \mathrm{Ps}_{\mathcal{D}}$ must be such that the subgame $\mathcal{D} \restriction \mathrm{A}$ satisfies $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Ps}_{\mathcal{D}} \setminus \mathrm{A}) = \emptyset$, for every quasi dominion pair $(\mathrm{Q}, \alpha) \in \mathrm{QD}_{\mathcal{D} \restriction \mathrm{A}}$. Observe that this requirement exactly mirrors Item (*ii*) in the definition of the g-stratified region function r. As already said before, we propose two instances of such a function. The first one computes the reachability set of an arbitrary position. The second one, instead, simply returns the positions of a minimal SCC. Clearly the escape of every quasi dominion in both the above subgames cannot reach positions outside those subgames.

Algorithm 6 implements the new successor function. Given the current state s and a compatible region pair (R, α) open in the whole input game, it produces a successor state $s^{\star} \triangleq (\mathsf{q}^{\star}, \mathsf{g}^{\star})$ in the dominion space. In particular, it first checks whether R is open also in the subgame $\mathcal{D}_s$ (Line 1). If this is the case, it assigns priority $p = \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ to region R and stores it in the new quasi dominion function $\mathsf{q}^{\star}$ (Line 2). The set A of positions of the new subgame is computed via the auxiliary function F (Line 3) and stored in the subgame function $\mathsf{g}^{\star}$ at priority $\mathsf{max}(\mathsf{rng}(\mathsf{q}^{\star} \restriction \mathrm{A}))$ (Line 4). If, on the other hand, R is closed in $\mathcal{D}_s$, a promotion is performed. The next priority $p^{\star}$ is set to the bep of R w.r.t. q for player $\overline{\alpha}$ in the entire game $\mathcal{D}$ (Line 5). Region R is, then, promoted to priority $p^{\star}$ and all the positions with priorities smaller than $p^{\star}$ in the current quasi dominion function q are reset (Line 6). Similarly, in the new subgame function $\mathsf{g}^{\star}$, the priorities of the same positions are set to $p^{\star}$ (Line 7). Also in this case, the correctness of this last operation follows from Proposition 1.

Algorithm 6: New successor function

signature $\downarrow : \succ \to \Delta \times \mathrm{Pr}$
function $s \downarrow (\mathrm{R}, \alpha)$
  let $(\mathsf{q}, \mathsf{g}) = s$ **in**
1 **if** $(\mathrm{R}, \alpha) \in \mathrm{Rg}^{-}_{\mathcal{D}_s}$ **then**
2 $\mathsf{q}^\star \leftarrow \mathsf{q}[\mathrm{R} \mapsto \mathsf{min}(\mathsf{rng}(\mathsf{g}))]$
3 $\mathrm{A} \leftarrow \mathsf{F}(\mathcal{D}_s \setminus \mathrm{R})$
4 $\mathsf{g}^\star \leftarrow \mathsf{g}[\mathrm{A} \mapsto \mathsf{max}(\mathsf{rng}(\mathsf{q}^\star \restriction \mathrm{A}))]$
  else
5 $p^\star \leftarrow \mathsf{bep}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}, \mathsf{q})$
6 $\mathsf{q}^\star \leftarrow \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{q}^{(\geq p^\star)}[\mathrm{R} \mapsto p^\star]$
7 $\mathsf{g}^\star \leftarrow \mathsf{g}[\mathsf{dom}\left(\mathsf{g}^{(<p^\star)}\right) \mapsto p^\star]$
8 **return** $(\mathsf{q}^\star, \mathsf{g}^\star)$

In conclusion, the priority-promotion mechanism extended with subgame decomposition forms a dominion space, as stated in the following theorem, whose proof is provided in the "Appendix".

Theorem 4 (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 6,* $\succ_{\mathcal{D}}$ *is the relation of Definition 7, and* $\Re_{\mathcal{D}}$ *and* $\downarrow_{\mathcal{D}}$ *are the functions computed by Algorithms 5 and 6 is a dominion space.*

6 Experimental evaluation

In this section we shall try to assess the effectiveness of the proposed approach. To this end, the priority promotion technique has been implemented in the tool PGSolver [31]. The tool, written in OCaml, collects implementations of several parity game solvers proposed in the literature and provides benchmarking tools, which can generate different forms of parity games. The available benchmarks divide into concrete and synthetic problems. 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, simple reachability problems, the Tower of Hanoi problems, and a fairness verification problem, the Elevator problem (see [31] for more details on this benchmarks). The *synthetic benchmarks* divide into randomly generated games and various families, corresponding to difficult cases, such as clique and ladder-like games, and worst cases for 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 few game transformation and decomposition techniques in the attempt to speed up the solution process. Indeed, those optimizations 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 [31]. Experiments were also conducted with different optimizations enabled and the results exhibit patterns similar to the ones emerging in the following experimental evaluation, even though solution times and the

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

Benchmark	Positions	Dom	Big	Str	Rec	SmPr	PP
Hanoi	7×10^5	1.4	1.4	†	1.8	†	**0.7**
Hanoi	2.1×10^6	4.4	4.4	†	5.7	†	**2.3**
Hanoi	6.3×10^6	21.4	21.4	‡	17.4	†	**7.0**
Elevator	1×10^5	1.0	1.0	†	0.8	12.8	**0.2**
Elevator	8.6×10^5	10.86	10.86	†	7.0	155.5	**2.0**
Elevator	7.7×10^6	†	‡	‡	‡	†	**19.8**
Lang. Incl.	3.7×10^5	6.6	6.6	†	3.0	78.4	**0.6**
Lang. Incl.	1.6×10^6	51.0	51.3	†	26.3	†	**3.6**
Lang. Incl.	5×10^6	†	‡	‡	145.5	‡	**16.5**
Ladder	4×10^6	†	‡	‡	35.0	130.5	**7.9**
Str. Imp.	4.5×10^6	81.0	82.8	†	71.0	‡	**57.0**
Clique	8×10^3	†	‡	†	†	†	**10.8**
MC. Lad.	7.5×10^6	†	‡	‡	**4.3**	‡	**4.3**
Rec. Lad.	5×10^4	†	‡	†	‡	‡	**62.8**
Jurdziński	4×10^4	†	†	188.2	†	93.2	**69.6**

gap among solvers may reduce considerably in some cases, depending on the benchmarks considered.

The algorithms considered in the experimentation are the Zielonka algorithm *Rec* [58], its two dominion decomposition variants, *Dom* [37,38] and *Big* [52], the strategy improvement algorithm *Str* [56],[1] the small progress measure algorithm *SmPr* [35], and the one of this article, PP (available at https://github.com/tcsprojects/pgsolver).[2]

6.1 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.[3] The parameter Positions refers to the number of positions in the games and the best results are emphasized in bold. The first three sections of the table, each comprising three instances of increasing size for the same benchmark, consider the concrete verification problems mentioned above. On all the instances of the Tower of Hanoi problem

[1] Note that the version of small-progress measure used in the experiments, which is included in the official release of PGSolver, is not the original one proposed by Jurdzinski, as it performs a progress interleaving for both players at once, instead of focusing on a single player like the original algorithm. According to the authors of PGSolver, this optimization allows for a non-trivial performance improvement.

[2] 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.

[3] The biggest instances in the table are generated by the following PGSolvercommands: `towersofhanoi 13`, `elevatorgame 8`, `langincl 500 100`, `laddergame 4000000`, `stratimprgen -pg friedmannsubexp 1000`, `modelcheckerladder 2500000`, `cliquegame 8000`, `recursiveladder 10001`, and `jurdzinskigame 100 100`.

most of the solvers perform reasonably well, except for *SmPr* and for *Str*. The Elevator problem, instead, proved to be very demanding, both in terms of time and memory, for all the solvers, except for our new algorithm and for *Dom*, which, however, could not solve the biggest instance within the time limit of 10 min. Our solver performs extremely well on both this benchmarks and on the instances of the Language Inclusion problem, whose biggest instance could be solved only by *Rec* among the other solvers. On the worst case benchmarks, PP performs quite well also on Ladder, Strategy Improvement, Clique, and Jurdziński games, all of which proved to be considerably difficult for all the other solvers. The Modelchecker game is a tie with *Rec*. On the Recursive Ladder game PP is the only algorithm that could solve the instance used in the experiments. However, the even instances of that game, unlike the odd ones, turn out to be very easy to solve by *Str*, which requires less than a second even for the instance with 5×10^5 positions. The reason is that those instances are completely won by player odd and, in addition, since *Str* looks for a winning strategy of player even, it immediately discovers that the initial even strategy cannot be improved further. On the other hand, the odd instances are completely won by player even and the algorithm requires a linear number of iterations to find the winning strategy.

The new solver exhibits the most consistent behavior overall on these benchmarks, significantly outperforming the other solvers considered in the experiments. Moreover, for all of them the priority promotion algorithm requires no promotions regardless of the input parameters, except for the Elevator problem, on which it performs only two. As a consequence, PP requires polynomial time on all the benchmarks reported in the table.

Table 3, instead, reports a comparison with the quasi-polynomial time algorithms proposed in [24] and [36]. We considered a preliminary C++ implementation of the former, *QPT*, made available by the authors themselves (the iOS executable can be obtained at https://cgi.csc.liv.ac.uk/~dominik/parity), and an OCaml implementation of the latter, *SPM*, recently included in PGSolver.[4] Despite the better worst case upper bound and the theoretical relevance of the result, neither algorithm could solve the benchmark instances reported in Table 2. Hence, we had to resort to much smaller instances. The comparison may not be particularly meaningful, given the differences in implementation, neither fair towards the solvers implemented in OCaml. It suggests, nonetheless, that the practical effectiveness of the two quasi-polynomial solvers, at least of the versions available at the time of writing, is very far from that of the exponential algorithms and, in particular, of PP. The table shows indeed a considerable gap, often of many orders of magnitude, which may only to some extent be explained by a possible lack of optimization.

6.2 Random games

Figure 4 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 20,000 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 s (3 min). 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 behavior of the solvers is typically highly variable even on games of the same size and priorities. To summarize 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 100 different games of the same size: for each size value n, we chose a

[4] At the time of writing, the *SPM* implementation is only provided in the development version of PGSolver, currently available at https://github.com/tcsprojects/pgsolver/tree/develop.

Table 3 Execution times (in seconds) on several benchmark families. Time out (†) is set to 600 s

Benchmark	Positions	QPT	SPM	PP
Hanoi	1.9×10^4	7.7	†	**0.0**
Hanoi	5.9×10^4	72.5	†	**0.0**
Elevator	2.7×10^3	13.0	†	**0.0**
Elevator	1.5×10^4	342.3	†	**0.0**
Elevator	1×10^5	†	†	**0.2**
Lang. Incl.	4.4×10^4	25.7	†	**0.0**
Lang. Incl.	7.8×10^4	71.8	†	**0.0**
Lang. Incl.	1.2×10^5	149.3	†	**0.1**
Ladder	1×10^4	121.1	†	**0.0**
Ladder	2×10^4	501.5	†	**0.0**
Str. Imp.	1.2×10^4	3.7	†	**0.0**
Str. Imp.	4.6×10^4	42.6	†	**0.0**
Str. Imp.	1×10^5	227.9	†	**0.1**
Clique	2×10^2	0.8	†	**0.0**
Clique	1×10^3	102.1	†	**0.1**
Clique	1.5×10^3	518.6	†	**0.4**
MC. Lad.	3×10^4	9.2	†	**0.0**
MC. Lad.	1.5×10^5	254.5	†	**0.0**
Rec. Lad.	5×10^3	13.1	†	**0.5**
Rec. Lad.	2.5×10^4	288.2	†	**13.8**
Jurdziński	7.5×10^3	54.6	†	**3.4**
Jurdziński	1.2×10^4	89.6	†	**9.6**
Jurdziński	1.9×10^4	320.4	†	**25.41**

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 performs 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. In these particular benchmarks, the additional memory required by PP amounts to 7 Mb on average for the biggest instances, while *Str* requires up to 83 Mb and *Rec* up to 422 Mb.

Table 4 reports on some experiments on random games with higher number of moves per position. The resulting games turn out to be much easier to solve for most the solvers, in particular for *Rec*, *Dom*, and PP. The benchmarks here are divided into 5 cluster with increasing number of positions from 10^4 to 10^5. Each cluster contains 120 games each, with number of priorities linear on the number of positions and 10–100 moves per position. On each row, the table reports the average time required for the solution over the 120 games of each cluster. The last row, instead, details the percentage of games on which the corresponding solver could not terminate before the timeout set to 60 s. The higher number of moves significantly increases the dimension of the regions computed by PP and, consequently, also the chances for it to find a closed one. Indeed, the number of promotions required by the algorithm on all

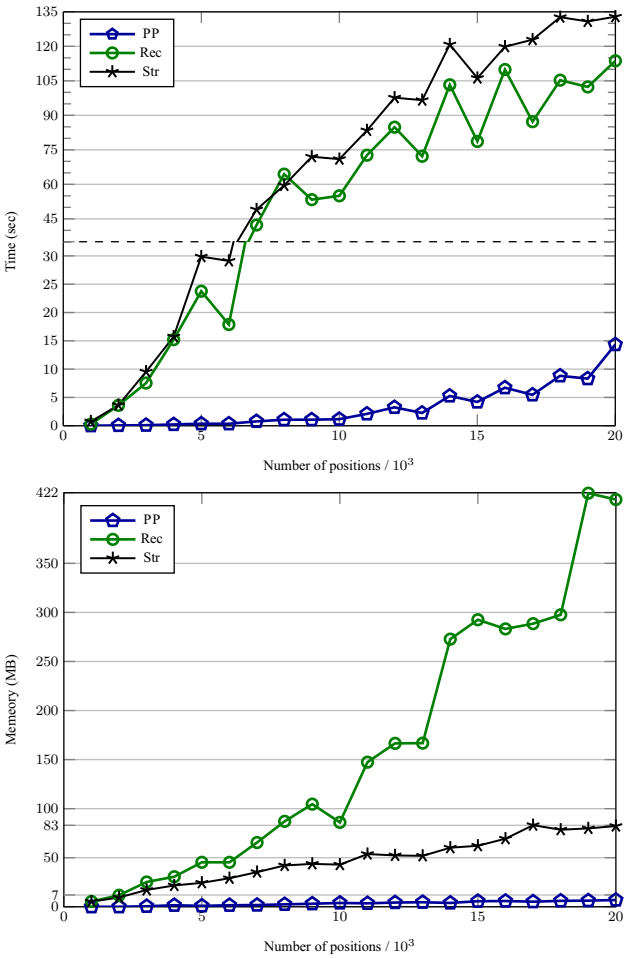


Fig. 4 Time and auxiliary memory on random games with 2 moves per position

Table 4 Average execution time (in seconds) on random games with 10 and 100 moves per position. Time out is set to 60 s

Positions	Dom	Big	Str	Rec	SmPr	PP
1×10^4	0.66	60.00	24.57	0.43	51.35	**0.12**
3×10^4	3.29	60.00	55.06	1.54	60.00	**0.46**
5×10^4	7.43	60.00	59.70	3.00	60.00	**0.97**
7×10^4	12.85	60.00	60.00	4.54	60.00	**1.50**
1×10^5	19.56	60.00	60.00	7.62	60.00	**2.28**
Time/Mem-out (%)	1	100	74	**0**	92	**0**

Table 5 Average execution time (in seconds) on random games with logarithmic number of priorities. Time out is set to 60 s

Positions	Dom	Big	Str	Rec	SmPr	PP
1×10^4	60.00	15.22	55.24	0.65	60.00	**0.14**
3×10^4	60.00	35.31	59.37	2.94	60.00	**0.40**
5×10^4	60.00	45.04	60.00	5.79	60.00	**1.39**
7×10^4	60.00	50.67	60.00	5.12	59.70	**0.79**
1×10^5	60.00	58.43	60.00	8.89	60.00	**1.83**
Timeout (%)	100	49	95.4	1	99.67	**0**

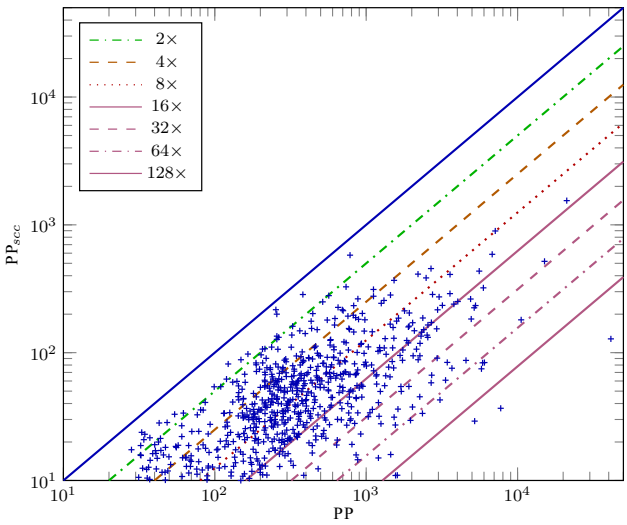


Fig. 5 Comparison between PP and PP_{scc} on random games with 50,000 positions

those games is typically below half a dozen. The whole solution time is almost exclusively due to a very limited number of attractors needed to compute the few regions contained in the games. The only other solvers that can easily solve all these games within 60 s are *Rec*, whose performance is only slightly worse than that of PP, and *Dom*, which could solve almost all of the games before the timeout.

Table 5 shows an experimental comparison on random games where the number of priorities grows logarithmically w.r.t. the number of positions. The relevance of these benchmarks stems from the fact that in practical applications, such as verification problems, the resulting encodings into parity games produce games with low number of priorities w.r.t. the number of positions. This number is usually connected to a measure of complexity of the temporal formula to verify, e.g., the alternation depth the fixpoint operators of the μ-calculus. In many cases, this number is bounded from above by a logarithmic function of the total number of positions in the game. As in the previous table, these benchmarks are divided into five clusters, according the number n of positions. Each cluster, in turn, contains 100 games with two moves per position and with number of priorities varying from $5 \log_2 n$ to $20 \log_2 n$. The results show that PP perform much better than the other solvers, being able to solve all of the benchmarks within the timeout, with *Rec* coming a close second.

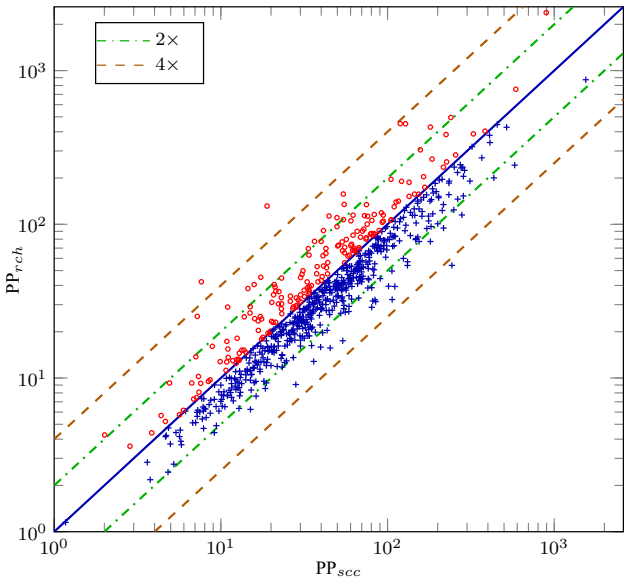


Fig. 6 Comparison between PP_{scc} and PP_{rch} on random games with 50,000 positions

6.3 PP with subgame decomposition

Finally, Fig. 5 compares the performance of the subgame decomposition version of PP with the original algorithm. Here, the decomposition schema has been realized by instantiating function F in Algorithm 6 with an SCC decomposition procedure that computes some minimal SCC in the current subgame. We refer to resulting procedure with PP_{scc}. In order to assess its effectiveness, we used a new pool of benchmarks that contains 740 games, each with 50,000 positions, 2 moves per positions and priorities varying from 8000 to 12,000. These games are much harder than the ones reported in Fig. 4 and have been specifically selected among random games whose solution requires PP more 30 s and up to more than 10 h to be solved. The results, drawn on a logarithmic scale, show that the decomposition technique, described in Sect. 6, does pay off significantly. It can run up to 128 times faster than the original version, reducing the solution time from several hours to a few seconds in some cases.

Very similar results can be obtained by instantiating function F in Algorithm 6 with a simpler reachability computation procedure that collects all the positions reachable from some arbitrary position in the current game. The resulting technique, called PP_{rch}, cannot, in general, provide as deep a decomposition as does the SCC-based decomposition. The additional overhead of the latter one is, however, usually higher. As shown in Fig. 6, the two algorithm are not comparable, in the sense that there is no clear winner between the two: on games with more structure the SCC decomposition usually performs better, being able to compute smaller subgames; on games with less structure, the reachability decomposition technique obtains sufficiently small subgames with less overhead.

7 Discussion

This article considers 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 propose a novel solution technique, based on a *priority-promotion mechanism*. Following this approach, a new solution algorithm is 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 is given. A lower bound for the worst-case is also presented in the form of a family of parity games on which the new technique exhibits an exponential behavior. 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 of 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 are conducted against concrete and synthetic problems. We compare 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, including the quasi-polynomial algorithms recently proposed in the literature. We also devise a quite general priority-promotion-based schema that can be instantiated with various game decomposition techniques, such as SCC-decomposition. Experiments comparing the original PP algorithm with two such instantiations reveal that a deep coupling of PP with decomposition techniques can be extremely helpful in cutting down solution time, often of several orders of magnitude.

These results suggest that the new approach is worth pursuing further. Therefore, we are currently investigating new and finer priority-promotion policies that try to minimize the number of region resets after a priority promotion.

It would also be interesting to investigate the applicability of the priority promotion approach to related problems, such as *prompt-parity games* [46] and similar conditions [14,29,30], and even in wider contexts like *mean-payoff games* [15,18] and *energy games* [12,13].

Appendix A: Proof of Theorem 1

To prove Theorem 1, we have to show that the three components $\mathcal{S}_\ni$, $\Re_\ni$, and $\downarrow_\ni$ of the structure $\mathcal{D}_\ni$ satisfy the properties required by Definition 2 of dominion space. We do this through the Lemmas 1, 2, and 3.

Lemma 1 (State Space) *The* PP *state space* $\mathcal{S}_\ni = \langle \mathrm{S}_\ni, \top_\ni, \prec_\ni \rangle$ *for a game* $\ni \in \mathcal{P}$ *is a well-founded partial order* w.r.t. $\prec_\ni$ *with designated element* $\top_\ni \in \mathrm{S}_\ni$.

Proof Since $\mathrm{S}_\ni$ is a finite set, to show that $\prec_\ni$ is a well-founded partial order on $\mathrm{S}_\ni$, it is enough to prove that it is simply a strict partial order on the same set, i.e., an irreflexive and transitive relation.

For the irreflexive property, by Item 3 of Definition 4, it is immediate to see that $s \not\prec_\ni s$, for all states $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_\ni$, since neither there exists a priority $q \in \mathsf{rng}(\mathsf{r})$ such that $\mathsf{r}^{-1}(q) \subset \mathsf{r}^{-1}(q)$ nor $p < p$.

For the transitive property, instead, consider three states $s_1 \triangleq (\mathsf{r}_1, p_1)$, $s_2 \triangleq (\mathsf{r}_2, p_2)$, $s_3 \triangleq (\mathsf{r}_3, p_3) \in \mathrm{S}_\ni$ for which $s_1 \prec_\ni s_2$ and $s_2 \prec_\ni s_3$ hold. Due to Items 3.a and 3.b of the same definition, four cases may arise.

- Item 3.a for both $s_1 \prec_{\mathcal{D}} s_2$ and $s_2 \prec_{\mathcal{D}} s_3$: there exist two priorities $q_1 \in \mathsf{rng}(\mathsf{r}_1)$ and $q_2 \in \mathsf{rng}(\mathsf{r}_2)$ with $q_1 \geq p_1$ such that ${\mathsf{r}_1}^{(>q_1)} = {\mathsf{r}_2}^{(>q_1)}$, ${\mathsf{r}_2}^{(>q_2)} = {\mathsf{r}_3}^{(>q_2)}$, $\mathsf{r}_2^{-1}(q_1) \subset \mathsf{r}_1^{-1}(q_1)$, and $\mathsf{r}_3^{-1}(q_2) \subset \mathsf{r}_2^{-1}(q_2)$. Let $q \triangleq \mathsf{max}\{q_1, q_2\} \geq p_1$. If $q = q_1 = q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. If $q = q_1 > q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = ({\mathsf{r}_2}^{(>q_2)})^{(>q)} = ({\mathsf{r}_3}^{(>q_2)})^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Finally, if $q = q_2 > q_1$ then ${\mathsf{r}_1}^{(>q)} = ({\mathsf{r}_1}^{(>q_1)})^{(>q)} = ({\mathsf{r}_2}^{(>q_1)})^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$. Moreover, $q \in \mathsf{rng}\big({\mathsf{r}_2}^{(>q_1)}\big) = \mathsf{rng}\big({\mathsf{r}_1}^{(>q_1)}\big) \subseteq \mathsf{rng}(\mathsf{r}_1)$. Summing up, it holds that $s_1 \prec_{\mathcal{D}} s_3$.
- Item 3.a for $s_1 \prec_{\mathcal{D}} s_2$ and Item 3.b for $s_2 \prec_{\mathcal{D}} s_3$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_1)$ with $q \geq p_1$ such that ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)}$ and $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$; moreover, $\mathsf{r}_2 = \mathsf{r}_3$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Consequently, $s_1 \prec_{\mathcal{D}} s_3$.
- Item 3.a for $s_2 \prec_{\mathcal{D}} s_3$ and Item 3.b for $s_1 \prec_{\mathcal{D}} s_2$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_2)$ with $q \geq p_2$ such that ${\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q)$; moreover, $\mathsf{r}_1 = \mathsf{r}_2$ and $p_1 < p_2$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$, $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$, $q \in \mathsf{rng}(\mathsf{r}_1)$, and $q > p_1$. Consequently, $s_1 \prec_{\mathcal{D}} s_3$.
- Item 3.b for both $s_1 \prec_{\mathcal{D}} s_2$ and $s_2 \prec_{\mathcal{D}} s_3$: $\mathsf{r}_1 = \mathsf{r}_2$, $\mathsf{r}_2 = \mathsf{r}_3$, $p_1 < p_2$, and $p_2 < p_3$. Hence, $\mathsf{r}_1 = \mathsf{r}_3$ and $p_1 < p_3$, which implies that $s_1 \prec_{\mathcal{D}} s_3$ also in this case.

To complete the proof, we need to show that $\top_{\mathcal{D}} \triangleq (\mathsf{pr}_{\mathcal{D}}, \mathsf{pr}(\mathcal{D}))$ belongs to $\mathrm{S}_{\mathcal{D}}$. Indeed, r is vacuously maximal above p, so Item 1.a holds. Item 1.c is trivially verified, since $\mathsf{r} = \mathsf{pr}_{\mathcal{D}}$. Obviously, Item 1.b also follows from the fact that $p = \mathsf{pr}(\mathcal{D}) = \mathsf{max}(\mathsf{rng}(\mathsf{pr}_{\mathcal{D}})) \in \mathsf{rng}\big(\mathsf{pr}_{\mathcal{D}}\big) = \mathsf{rng}(\mathsf{r})$. Finally, $\mathsf{pr}_{\mathcal{D}}$ is a region function as, for all priorities $q \in \mathsf{rng}\big(\mathsf{pr}_{\mathcal{D}}\big)$ with $\alpha \triangleq q \bmod 2$, it holds that $\mathsf{pr}_{\mathcal{D}}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{pr}_{\mathcal{D}}}}$ is an α-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{pr}_{\mathcal{D}}}$, if non-empty. Indeed, the set $\mathsf{pr}_{\mathcal{D}}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{pr}_{\mathcal{D}}}}$ can only contain positions of priority p, which is the maximal one in the corresponding subgame. Therefore, player α has an obvious strategy that forces every infinite play inside this set to be winning for it. □

Lemma 2 (Query Function) *The function $\mathfrak{R}_{\mathcal{D}}$ is a query function,* i.e., for all states $s \in \mathrm{S}_{\mathcal{D}}$, it holds that *(1)* $\mathfrak{R}_{\mathcal{D}}(s) \in \mathrm{QD}_{\mathcal{D}}$ and *(2)* if $\mathfrak{R}_{\mathcal{D}}(s) \in \mathrm{QD}^{-}_{\mathcal{D}}$ then $s \succ_{\mathcal{D}} \mathfrak{R}_{\mathcal{D}}(s)$.

Proof Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_{\mathcal{D}}$ be a state and $(\mathrm{R}, \alpha) \triangleq \mathfrak{R}_{\mathcal{D}}(s)$ the pair of a set of positions $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}}$ and a player $\alpha \in \{0, 1\}$ obtained by computing the function $\mathfrak{R}_{\mathcal{D}}$ on s. Due to Line 1 of Algorithm 3, it follows that $\alpha \equiv_2 p$. By Item 1.b, it holds that $p \in \mathsf{rng}(\mathsf{r})$. Thus, by Item 1.a and the definition of region function, we have that $\mathsf{r}^{-1}(p)$ is an α-region in $\mathcal{D}^{\leq p}_{\mathsf{r}}$, the latter being equal to $\mathcal{D}_s$. Now, by Line 2 of Algorithm 3 and Proposition 2, where we assume $\mathrm{R}^{\star} \triangleq \mathsf{r}^{-1}(p)$ and $\mathcal{D} \triangleq \mathcal{D}_s$, it follows that $\mathrm{R} \supseteq \mathsf{r}^{-1}(p)$ is an α-region in $\mathcal{D}_s$, i.e., $(\mathrm{R}, \alpha) \in \mathrm{Rg}_{\mathcal{D}_s}$, and, so a quasi dominion in $\mathcal{D}$, i.e., $(\mathrm{R}, \alpha) \in \mathrm{QD}_{\mathcal{D}}$. In addition, R is α-maximal in $\mathcal{D}_s$. Consequently, $s \succ_{\mathcal{D}} (\mathrm{R}, \alpha)$, since all requirements of Definition 5 are satisfied. □

Lemma 3 (Successor Function) *The function $\downarrow_{\mathcal{D}}$ is an successor function,* i.e., *for all states $s \in \mathrm{S}_{\mathcal{D}}$ and region pairs $(\mathrm{R}, \alpha) \in \mathrm{Rg}^{-}_{\mathcal{D}}$ with $s \succ_{\mathcal{D}} (\mathrm{R}, \alpha)$, it holds that (1) $s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha) \in \mathrm{S}_{\mathcal{D}}$ and (2) $s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha) \prec_{\mathcal{D}} s$.*

Proof Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_{\mathcal{D}}$ be a state, $(\mathrm{R}, \alpha) \in \mathrm{Rg}^{-}_{\mathcal{D}}$ an open region pair in $\mathcal{D}$ compatible with s, and $s^{\star} = (\mathsf{r}^{\star}, p^{\star}) \triangleq s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha)$ the result obtained by computing the function $\downarrow_{\mathcal{D}}$ on s and (R, α). Due to Item 1 of Definition 4, we have that *(1)* r is maximal above p and *(2)* $\mathsf{r}^{(<p)} \subseteq {\mathsf{pr}_{\mathcal{D}}}^{(<p)}$. Moreover, by Item 1 of Definition 5, it holds that $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}_s}$, which implies *(3)* $\mathsf{dom}\big(\mathsf{r}^{(>p)}\big) \cap \mathrm{R} = \emptyset$.

On the one hand, suppose that R is α-open in $\mathfrak{D}_s$, i.e., $(R, \alpha) \in \mathrm{Rg}^{-}_{\mathfrak{D}_s}$. By Lines 1–3 of Algorithm 4, we have that (*4*) $\mathsf{r}^\star = \mathsf{r}[R \mapsto p]$ and (*5*) $p^\star = \mathsf{max}(\mathsf{rng}(\mathsf{r}^{\star(<p)}))$. In addition, by Item 2 of Definition 5, it holds that (*6*) R is α-maximal in $\mathfrak{D}_s$ and (*7*) $\mathsf{r}^{-1}(p) \subseteq R$. Now, by Point (*5*), it immediately follows that (*8*) $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$, (*9*) $p^\star < p$, and (*10*) there is no priority $q \in \mathsf{rng}(\mathsf{r}^\star)$ such that $p^\star < q < p$. Also, by Points (*4*), (*3*) and (*7*), we have that $\mathsf{r}^{\star(>p)} = \mathsf{r}^{(>p)}$, $\mathsf{r}^{\star-1}(p) = R$, and $\mathsf{r}^{\star(<p)} \subseteq \mathsf{r}^{(<p)}$. Thus, by Points (*1*), (*6*), (*9*), and (*10*), it holds that (*11*) $\mathsf{r}^\star$ is maximal above $p^\star$. Moreover, by Points (*2*) and (*9*), we derive that (*12*) $\mathsf{r}^{\star(<p^\star)} \subseteq \mathsf{pr}_{\mathfrak{D}}{}^{(<p^\star)}$. Finally, to prove that (*13*) $\mathsf{r}^\star$ is a region function, we need to show that $\mathsf{r}^{\star-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}_{\mathsf{r}^\star}^{\trianglelefteq q}}$, if non-empty, is a β-region in the subgame $\mathfrak{D}_{\mathsf{r}^\star}^{\trianglelefteq q}$, for all priorities $q \in \mathsf{rng}(\mathsf{r}^\star)$ with $\beta \triangleq q \bmod 2$. Indeed, if $q > p$, by Point (*1*), it holds that $\mathsf{r}^{\star-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}_{\mathsf{r}^\star}^{\trianglelefteq q}} = \mathsf{r}^{-1}(q) \neq \emptyset$. Hence, the property follows directly from the fact that r is a region function. If $q = p$, again by Point (*1*), we have that $\mathsf{r}^{\star-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}_{\mathsf{r}^\star}^{\trianglelefteq q}} = R \neq \emptyset$. So, the property is ensured by the hypothesis that R is an α-region. For the last case $q < p$, the property follows by Point (*12*) and the observation that, if non-empty, the set $\mathsf{r}^{\star-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}_{\mathsf{r}^\star}^{\trianglelefteq q}} \subseteq \mathsf{pr}_{\mathfrak{D}}^{-1}(q)$ only contains positions of priority q that necessarily form a region of the corresponding parity. Summing up, Points (*13*), (*11*), (*8*), and (*12*) ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}_{\mathfrak{D}}$. At this point, it remains just to show that $(\mathsf{r}^\star, p^\star) \prec_{\mathfrak{D}} (\mathsf{r}, p)$. By Point (*7*), two cases may arise. If $\mathsf{r}^{-1}(p) \subset R$, the thesis follows from Item 3.a of Definition 4, where the priority q is set to p. On the contrary, if $R = \mathsf{r}^{-1}(p)$, by Point (*4*), we have that $\mathsf{r}^\star = \mathsf{r}$. Therefore, due to Point (*9*), the thesis is derived from Item 3.b of the same definition.

On the other hand, suppose that R is α-closed in $\mathfrak{D}_s$, i.e., $(R, \alpha) \notin \mathrm{Rg}^{-}_{\mathfrak{D}_s}$. By Lines 1, 4, and 5 of Algorithm 4, we have that (*14*) $p^\star = \mathsf{bep}^{\overline{\alpha}}_{\mathfrak{D}}(R, \mathsf{r})$ and (*15*) $\mathsf{r}^\star = \mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{r}^{(\geq p^\star)}[R \mapsto p^\star]$. It is not hard to see that, by Points (*14*) and (*7*), the definition of bep and the fact that R is closed, we have that (*16*) $p^\star > p$. Now, by Points (*15*) and (*3*), it follows that $\mathsf{r}^{\star(>p^\star)} = \mathsf{r}^{(>p^\star)}$, $\mathsf{r}^{\star-1}(p^\star) = \mathsf{r}^{-1}(p^\star) \cup R$, and (*18*) $\mathsf{r}^{\star(<p^\star)} \subseteq \mathsf{pr}_{\mathfrak{D}}{}^{(<p^\star)}$. Consequently, (*19*) $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$. Moreover, due to Points (*1*) and (*16*), we have that (*20*) $\mathsf{r}^\star$ is maximal above $p^\star$. Finally, the proof that (*21*) $\mathsf{r}^\star$ is a region function easily follows by observing that $\mathsf{r}^{-1}(p^\star) \cup R$ is an α-region, due to Proposition 1. Summing up, Points (*21*), (*20*), (*19*), and (*18*) ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}_{\mathfrak{D}}$. At this point, as for the previous case, it remains to show that $(\mathsf{r}^\star, p^\star) \prec_{\mathfrak{D}} (\mathsf{r}, p)$. This fact easily follows from Item 3.a of Definition 4, where the priority q is set to $p^\star$, since, by Points (*3*) and (*16*), we have that $\mathsf{r}^{-1}(p^\star) \cap R = \emptyset$, so $\mathsf{r}^{-1}(p^\star) \subset \mathsf{r}^{-1}(p^\star) \cup R = \mathsf{r}^{\star-1}(p^\star)$. □

Appendix B: Proof of Theorem 4

Similarly to the previous appendix, to prove Theorem 4, we have to show that the three components $\mathcal{S}_{\mathfrak{D}}$, $\Re_{\mathfrak{D}}$, and $\downarrow_{\mathfrak{D}}$ of the structure $\mathcal{D}_{\mathfrak{D}}$ satisfy the properties required by Definition 6 of dominion space. We do this through the Lemmas 4, 5, and 6.

Lemma 4 (State Space) *The generalized* PP *state space* $\mathcal{S}_{\mathfrak{D}} = \langle \mathrm{S}_{\mathfrak{D}}, \top_{\mathfrak{D}}, \prec_{\mathfrak{D}} \rangle$ *for a game* $\mathfrak{D} \in \mathcal{P}$ *is a well-founded partial order* w.r.t. $\prec_{\mathfrak{D}}$ with designated element $\top_{\mathfrak{D}} \in \mathrm{S}_{\mathfrak{D}}$.

Proof To prove that $\prec_{\mathfrak{D}}$ is a strict partial order on $\mathrm{S}_{\mathfrak{D}}$, we can follow the same approach used in the proof of Theorem 1, where the pair $(\mathsf{q} \cap \mathsf{g}, \mathsf{min}(\mathsf{rng}(\mathsf{g})))$ plays the same role of the state (r, p) in the PP algorithm. Therefore, to complete the proof, we only need to show that $\top_{\mathfrak{D}} \triangleq (\mathsf{pr}_{\mathfrak{D}}, \varnothing[\mathrm{Ps}_{\mathfrak{D}} \mapsto \mathsf{pr}(\mathfrak{D})])$ belongs to $\mathrm{S}_{\mathfrak{D}}$. The function $\mathsf{q} \triangleq \mathsf{pr}_{\mathfrak{D}}$ is trivially

a quasi dominion function, since $\mathsf{pr}_{\mathcal{D}}^{-1}(p)$ is an α-quasi dominion in the original game $\mathcal{D}$ as it only contains positions of the same priority p of parity α, for all $p \in \mathsf{rng}(\mathsf{pr}_{\mathcal{D}})$ with $\alpha \triangleq p \bmod 2$. To prove that $\mathsf{g} \triangleq \varnothing[\mathrm{Ps}_{\mathcal{D}} \mapsto \mathsf{pr}(\mathcal{D})]$ is a subgame function, first observe that Item (*i*) of the corresponding definition holds due to the fact that $\mathsf{rng}(\mathsf{g}) = \{\mathsf{pr}(\mathcal{D})\}$ and $\mathsf{pr}(\mathcal{D}) \in \mathsf{rng}(\mathsf{pr}_{\mathcal{D}})$. Moreover, $\mathsf{dom}(\mathsf{g}^{(\leq \mathsf{pr}(\mathcal{D}))}) = \mathrm{Ps}_{\mathcal{D}}$, so, $\mathcal{D}_{\mathsf{g}}^{\leq \mathsf{pr}(\mathcal{D})} = \mathcal{D}$ is clearly a game. Consequently, Item (*ii.a*) holds as well. Finally, Item (*ii.b*) is vacuously verified, since there is no priority $p \in \mathsf{rng}(\mathsf{g})$ with $p > \mathsf{pr}(\mathcal{D})$. We can now consider the three points in Item 1 of Definition 6. Point (*a*) vacuously holds, since, as already observed, there is no priority in g greater than $\mathsf{pr}(\mathcal{D})$. Now, let $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. First notice that $\mathsf{r}^{-1}(\mathsf{pr}(\mathcal{D})) = \mathsf{pr}_{\mathcal{D}}^{-1}(\mathsf{pr}(\mathcal{D}))$ is a region in $\mathcal{D}_{\top_{\mathcal{D}}} = \mathcal{D}_{\mathsf{g}}^{\leq \mathsf{pr}(\mathcal{D})} = \mathcal{D}$, since it is a quasi dominion whose escape positions have the maximal priority in the game. Moreover, $X = \mathrm{Ps}_{\mathcal{D}} \setminus (\mathsf{dom}(\mathsf{g}^{(\leq \mathsf{pr}(\mathcal{D}))}) \cup \mathsf{dom}(\mathsf{r})) = \mathrm{Ps}_{\mathcal{D}} \setminus \mathrm{Ps}_{\mathcal{D}} = \emptyset$. Consequence, Point (*b*) also holds. Finally, Point (*c*) is implied by the fact that $\mathsf{pr}_{\mathcal{D}} \uplus \mathsf{r} = \mathsf{pr}_{\mathcal{D}} \uplus (\mathsf{q} \cap \mathsf{g}) = \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{q} = \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{pr}_{\mathcal{D}} = \mathsf{pr}_{\mathcal{D}} = \mathsf{q}$. □

Lemma 5 (Query Function) *The function $\mathfrak{R}_{\mathcal{D}}$ is a query function,* i.e., for all states $s \in \mathrm{S}_{\mathcal{D}}$, it holds that (*1*) $\mathfrak{R}_{\mathcal{D}}(s) \in \mathrm{QD}_{\mathcal{D}}$ and (*2*) if $\mathfrak{R}_{\mathcal{D}}(s) \in \mathrm{QD}_{\mathcal{D}}^{-}$ then $s \succ_{\mathcal{D}} \mathfrak{R}_{\mathcal{D}}(s)$.

Proof Let $s \triangleq (\mathsf{q}, \mathsf{g}) \in \mathrm{S}_{\mathcal{D}}$ be a state and $(\mathrm{R}, \alpha) \triangleq \mathfrak{R}_{\mathcal{D}}(s)$ the pair of a set of positions $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}}$ and a player $\alpha \in \{0, 1\}$ obtained by computing the function $\mathfrak{R}_{\mathcal{D}}$ on s. Moreover, assume $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. First observe that Item 1.c implies $\mathsf{rng}(\mathsf{g}) \subseteq \mathsf{rng}(\mathsf{q})$. Indeed, suppose by contradiction that there exists a priority $p \in \mathsf{rng}(\mathsf{g}) \setminus \mathsf{rng}(\mathsf{q})$. Clearly, $p \notin \mathsf{rng}(\mathsf{r})$. Moreover, $p \in \mathsf{rng}(\mathsf{g}) \subseteq \mathsf{rng}(\mathsf{pr}_{\mathcal{D}})$, due to Item (*i*) of the definition of subgame function applied to g. Therefore, $p \in \mathsf{rng}(\mathsf{pr}_{\mathcal{D}} \uplus \mathsf{r}) = \mathsf{rng}(\mathsf{q})$, but this is obviously impossible. Now, by Line 1 of Algorithm 5 and the definition of the subgame $\mathcal{D}_s$, we have that $\mathsf{q}^{-1}(p) \cap \mathrm{Ps}_{\mathcal{D}_s} = \mathsf{q}^{-1}(p) \cap \mathsf{dom}(\mathsf{g}^{(\leq p)}) = \mathsf{q}^{-1}(p) \cap \mathsf{g}^{-1}(p) = \mathsf{r}^{-1}(p) \neq \emptyset$. Thus, due to Line 2 of the same algorithm and Item 1.b, it holds that $\mathsf{r}^{-1}(p)$ is an α-region in $\mathcal{D}_s$. Finally, by Line 3 and Proposition 2, R is α-maximal in $\mathcal{D}_s$. Consequently, both Point (*1*) and the requirements of Definition 5 are satisfied. □

Lemma 6 (Successor Function) *The function $\downarrow_{\mathcal{D}}$ is an successor function,* i.e., *for all states $s \in \mathrm{S}_{\mathcal{D}}$ and quasi dominion pairs $(\mathrm{R}, \alpha) \in \mathrm{QD}_{\mathcal{D}}^{-}$ with $s \succ_{\mathcal{D}} (\mathrm{R}, \alpha)$, it holds that (1) $s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha) \in \mathrm{S}_{\mathcal{D}}$ and (2) $s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha) \prec_{\mathcal{D}} s$.*

Proof Let $s \triangleq (\mathsf{q}, \mathsf{g}) \in \mathrm{S}_{\mathcal{D}}$ be a state, $(\mathrm{R}, \alpha) \in \mathrm{QD}_{\mathcal{D}}^{-}$ an open quasi dominion pair in $\mathcal{D}$ compatible with s, and $s^{\star} = (\mathsf{q}^{\star}, \mathsf{g}^{\star}) \triangleq s \downarrow_{\mathcal{D}} (\mathrm{R}, \alpha)$ the result obtained by computing the function $\downarrow_{\mathcal{D}}$ on s and (R, α). Moreover, assume $p = \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ and $\mathsf{r} = \mathsf{q} \cap \mathsf{g}$. Two cases may be arises: R is either α-open or α-closed in $\mathcal{D}_s$, i.e., $(\mathrm{R}, \alpha) \in \mathrm{Rg}_{\mathcal{D}_s}^{-}$ or $(\mathrm{R}, \alpha) \notin \mathrm{Rg}_{\mathcal{D}_s}^{-}$, respectively.

In the first case, by Lines 1-4 of Algorithm 6, we have that (*1*) $\mathsf{q}^{\star} = \mathsf{q}[\mathrm{R} \mapsto p]$, (*2*) $\mathrm{A} = \mathsf{F}(\mathcal{D}_s \setminus \mathrm{R})$, and (*3*) $\mathsf{g}^{\star} = \mathsf{g}[\mathrm{A} \mapsto \mathsf{max}(\mathsf{rng}(\mathsf{q}^{\star} \restriction \mathrm{A}))]$. By Definition 7, it holds that $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}_s}$. Thus, (*4*) $\mathrm{R} \subseteq \mathsf{g}^{-1}(p)$, i.e.R does not contain positions with priority higher than p. As consequence of Points (*1*) and (*4*), we have that (*5*) $\mathsf{q}^{\star(>p)} = \mathsf{q}^{\star(>p)}$, (*6*) $\mathsf{q}^{\star(<p)} \subseteq \mathsf{q}^{(<p)}$, and (*7*) $\mathsf{q}^{\star -1}(p) = \mathrm{R} \cup \mathsf{q}^{-1}(p)$.

By Point (*5*) and the assumption on q, we have that $\mathsf{q}^{\star -1}(p')$ is an α'-quasi dominion in $\mathcal{D}$, for all priorities $p \in \mathsf{rng}(q)$ with $p' > p$ and $\alpha' \triangleq p' \bmod 2$. Moreover, by Item 1.c of Definition 6, for every priority $p \in \mathsf{rng}(q)$ with $p' < p$ and $\alpha' \triangleq p' \bmod 2$, it holds that (*8*) $\mathsf{q}^{-1}(p') \subseteq \mathsf{pr}_{\mathcal{D}}^{-1}(p')$, so, by Point (*6*), $\mathsf{q}^{\star -1}(p')$ is an α'-quasi dominion in $\mathcal{D}$ as well. Again by Item 1.c, we have that $\mathsf{q}^{-1}(p)$ is the union of $\mathsf{r}^{-1}(p)$ with a set $\mathrm{P} \subseteq \mathsf{pr}_{\mathcal{D}}^{-1}(p) \setminus \mathsf{g}^{-1}(p)$. Moreover, (*9*) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$, due to Definition 7. Thus, by Point (*7*), it holds that

(*10*) $\mathsf{q}^{\star\,-1}(p) = \mathrm{R} \cup \mathsf{q}^{-1}(p) = \mathrm{R} \cup \mathrm{P} \cup \mathsf{r}^{-1}(p) = \mathrm{R} \cup \mathrm{P}$, i.e., $\mathsf{q}^{\star\,-1}(p)$ is the union of an α-quasi dominion in $\mathcal{D}$ with some set of positions of priority p having the same parity. Consequently, also $\mathsf{q}^{\star\,-1}(p)$ is a α-quasi dominion in $\mathcal{D}$. In conclusion, we have that (*11*) $\mathsf{q}^\star$ is a quasi-dominion function.

By Point (*2*) and the assumption on the function F, we know that (*12*) $\emptyset \neq \mathrm{A} \subseteq \mathcal{D}_s \setminus \mathrm{R} = \mathsf{dom}\big(\mathsf{g}^{\leq p}\big) \setminus \mathrm{R}$ induces a subgame $\mathcal{D}{\restriction}\mathrm{A}$ of $\mathcal{D}_s \setminus \mathrm{R}$ such that $\mathsf{esc}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{Ps}_{\mathcal{D}} \setminus \mathrm{A}) = \emptyset$, for every quasi dominion pair $(\mathrm{Q}, \alpha) \in \mathrm{QD}_{\mathcal{D}{\restriction}\mathrm{A}}$. In addition, it is easy to see that $\mathsf{pr}^{-1}_{\mathcal{D}}(p) \cap \mathrm{Ps}_{\mathcal{D}_s} \subseteq \mathrm{R}$. Indeed, if by contradiction this was not the case, by definition of the subgame $\mathcal{D}_s$ and Point (*9*), there would be a position $v \in \mathsf{pr}^{-1}_{\mathcal{D}}(p) \cap \mathsf{g}^{-1}(p)$ such that $v \notin \mathsf{r}^{-1}(p)$. Therefore, due to Item 1.c of Definition 6, $v \in \mathsf{q}^{-1}(p)$. However, $\mathsf{r} = \mathsf{q} \cap \mathsf{g}$, so, $v \in \mathsf{q}^{-1}(p) \cap \mathsf{q}^{-1}(p) = \mathsf{r}^{-1}(p)$, which is impossible. As a consequence of this inclusion together with Point (*12*), we have that (*13*) $p^\star = \mathsf{max}(\mathsf{rng}(\mathsf{q}^\star \restriction \mathrm{A})) < p$. Now, due to Points (*3*) and (*13*), it holds that (*14*) $\mathsf{g}^{\star\,(>p)} = \mathsf{g}^{(>p)}$, (*15*) $\mathsf{g}^{\star\,-1}(p) = \mathsf{g}^{-1}(p) \setminus \mathrm{A}$, (*16*) $\mathsf{dom}\Big(\mathsf{g}^{\star\,(<p)}\Big) = \mathsf{g}^{\star\,-1}(p^\star) = \mathrm{A}$, and (*17*) $\mathsf{dom}\Big(\mathsf{g}^{\star\,(<p^\star)}\Big) = \emptyset$. Observe that, by Point (*14*), g and $\mathsf{g}^\star$ only differ on the priorities p and $p^\star$. In particular, $\mathsf{rng}(\mathsf{g}^\star) = \mathsf{rng}(\mathsf{g}) \cup \{p^\star\}$. Thus, by Item (*i*) of the definition of subgame function and Points (*8*) and (*13*), we have that $\mathsf{rng}(\mathsf{g}^\star) \subseteq \mathsf{rng}\big(\mathsf{pr}_{\mathcal{D}}\big)$. By Points (*15*) and (*16*), (*18*) $\mathcal{D}^{\leq p}_{\mathsf{g}^\star} = \mathcal{D}^{\leq p}_{\mathsf{g}}$ is obviously a game. Moreover, due to Point (*4*), it holds that (*19*) $\mathsf{dom}\Big(\mathsf{g}^{\star\,(\leq p^\star)}\Big) = \mathrm{A} \subset \mathsf{dom}\big(\mathsf{g}^{\star\,(\leq p)}\big)$. Finally, $\mathcal{D}^{\leq p^\star}_{\mathsf{g}^\star}$ is a game, by Point (*12*). Summing up, it follows that (*20*) $\mathsf{g}^\star$ is a subgame function.

We can now prove the tree parts in which Item 1 of Definition 6 applied to $(\mathsf{q}^\star, \mathsf{g}^\star)$ is split. First notice that, since $\mathsf{r} = \mathsf{q} \cap \mathsf{g}$, we have that (*21*) $\mathsf{r}^{\star\,(>p)} = \mathsf{r}^{(>p)}$, due to Points (*5*) and (*14*). Moreover, (*22*) $\mathsf{r}^{\star\,-1}(p^\star) \subseteq \mathsf{pr}^{-1}_{\mathcal{D}}(p^\star)$, by Point (*8*), and (*23*) $\mathsf{r}^{\star\,(<p^\star)} = \varnothing$, by Point (*17*). Finally, by Points (*10*), (*13*), and (*15*) and the observation that $\mathrm{P} \cap \mathsf{g}^{-1}(p) = \emptyset$, it holds that (*24*) $\mathsf{r}^{\star\,-1}(p) = \mathrm{R}$. At this point, Item 1.a of the definition of state space follows from Points (*21*) and (*24*), since R is α-maximal in $\mathcal{D}_s$ due to Definition 7. By putting together Points (*18*)–(*23*) with Point (*12*), also Item 1.b can be derived. To prove that Item 1.c, i.e., $\mathsf{q}^\star = \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{r}^\star$, one can use Points (*5*) and (*21*) for the priorities greater than p, Points (*10*) and (*24*) for the priority p, and Point (*8*) for the priorities smaller than p.

In the end, by Points (*11*) and (*20*), and the fact that Item 1 of Definition 6 holds on $(\mathsf{q}^\star, \mathsf{g}^\star)$, it follows that $(\mathsf{q}^\star, \mathsf{g}^\star) \in \mathrm{S}_{\mathcal{D}}$.

To conclude this case of the proof, we need just to show that $(\mathsf{q}^\star, \mathsf{g}^\star) \prec_{\mathcal{D}} (\mathsf{q}, \mathsf{g})$. If $\mathrm{R} \subseteq \mathsf{q}^{-1}(p)$, by Point (*1*) and (*13*), we have that $\mathsf{q}^\star = \mathsf{q}$ and $p^\star < p$. Thus, the thesis is derived from Item 3.b of Definition 6. If, on the other hand, $\mathrm{R} \setminus \mathsf{q}^{-1}(p) \neq \emptyset$, due to Point (*5*) and (*7*), the thesis follows from Item 3.a of the same definition.

Consider now the case in which the set R is α-closed in $\mathcal{D}_s$. By Lines 1, 5–7 of Algorithm 6, we have that (*25*) $p^\star = \mathsf{bep}^{\overline{\alpha}}_{\mathcal{D}}(\mathrm{R}, \mathsf{q})$, (*26*) $\mathsf{q}^\star = \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{q}^{(\geq p^\star)}[\mathrm{R} \mapsto p^\star]$, and (*27*) $\mathsf{g}^\star = \mathsf{g}[\mathsf{dom}\Big(\mathsf{g}^{(<p^\star)}\Big) \mapsto p^\star]$. Due to Point (*25*), the definition of best escape priority, and the fact that R is α-closed in $\mathcal{D}_s$, it is not hard to see that (*28*) $p^\star > p$. Moreover, due to Item (*ii*) of the definition of g stratified region function, it holds that (*29*) $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$, since the moves escaping from R can only reach positions in $\mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$.

Now, by Point (*27*), it follows that (*30*) $\mathsf{g}^{\star\,(>p^\star)} = \mathsf{g}^{(>p^\star)}$, (*31*) $\mathsf{g}^{\star\,(<p^\star)} = \emptyset$, and (*32*) $\mathsf{dom}\Big(\mathsf{g}^{(\leq p^\star)}\Big) = \mathsf{dom}\Big(\mathsf{g}^{(\leq p^\star)}\Big)$. Therefore, it is immediate to see that (*33*) $\mathsf{g}^\star$ is a subgame function.

Moreover, by Point (*26*), it holds that (*34*) $\mathsf{q}^{\star\,(>p^\star)} = \mathsf{q}^{(>p^\star)}$, (*35*) $\mathsf{q}^{\star\,-1}(p^\star) = \mathsf{q}^{-1}(p^\star) \cup \mathrm{R}$, and (*36*) $\mathsf{q}^{\star\,(<p^\star)} \subseteq \mathsf{pr}_{\mathcal{D}}{}^{(<p^\star)}$. Consequently, (*37*) $\mathsf{r}^{\star\,(>p^\star)} = \mathsf{r}^{(>p^\star)}$, by

Points (*30*) and (*34*), and (*38*) $\mathsf{r}^{\star(<p^\star)} = \emptyset$, by Point (*31*). In addition, due to Points (*32*) and (*36*), together with Item (*ii.b*) of the definition of subgame function and Proposition 1, we have that (*39*) $\mathsf{r}^{\star-1}(p^\star) = \mathsf{q}^{\star-1}(p^\star) \cap \mathsf{g}^{\star-1}(p^\star) = (\mathsf{q}^{-1}(p^\star) \cup \mathrm{R}) \cap \mathsf{g}^{\star-1}(p^\star) = \mathsf{r}^{-1}(p^\star) \cup \mathrm{R}$ is an α-region in $\mathcal{D}_{s^\star}$. At this point, Item 1 of Definition 6 follows by simply verifying that all requirements are satisfied.

Because of Points (*34*)–(*36*), to prove that (*40*) $\mathsf{q}^\star$ is a quasi-dominion function, it is enough to observe that $\mathsf{q}^{\star-1}(p^\star)$ is the union of $\mathsf{r}^{\star-1}(p^\star))$ with a set $\mathrm{P} \subseteq \mathsf{pr}_{\mathcal{D}}^{-1}(p^\star)) \setminus \mathsf{g}^{\star-1}(p^\star))$. Indeed, $\mathsf{q}^{\star-1}(p^\star)$ is an α-quasi dominion in $\mathcal{D}$, being the union of an α-region in $\mathcal{D}_{s^\star}$ and a set of positions having priority $p^\star$).

In the end, by Points (*33*) and (*40*), and the fact that Item 1 of Definition 6 holds on $(\mathsf{q}^\star, \mathsf{g}^\star)$, it follows that $(\mathsf{q}^\star, \mathsf{g}^\star) \in \mathrm{S}_{\mathcal{D}}$.

To conclude, as for the previous case, it remains to show that $(\mathsf{q}^\star, \mathsf{g}^\star) \prec_{\mathcal{D}} (\mathsf{q}, \mathsf{g})$. This fact easily follows from Item 3.a of Definition 6, where the priority p is set to $p^\star$. Indeed, by Definition 7, we have that $\emptyset \neq \mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}_s}$, so, $\mathsf{q}^{-1}(p^\star) \cap \mathrm{R} = \emptyset$, since $p^\star > p$ as stated in Point (*28*). Therefore, by Point (*31*), it follows that $\mathsf{q}^{-1}(p^\star) \subset \mathsf{q}^{-1}(p^\star) \cup \mathrm{R} = \mathsf{q}^{\star-1}(p^\star)$. □

References

1. Agrawal M, Kayal N, Saxena N (2004) PRIMES is in P. Ann Math 160(2):781–793
2. Alur R, Henzinger TA, Kupferman O (2002) Alternating-time temporal logic. J ACM 49(5):672–713
3. Apt K, Grädel E (2011) Lectures in game theory for computer scientists. Cambridge University Press, Cambridge
4. Benerecetti M, Dell'Erba D, Mogavero F (2016) Solving parity games via priority promotion. In Computer aided verification'16, LNCS 9780 (Part II). Springer, New York, pp 270–290
5. Benerecetti M, Mogavero F, Murano A (2013) Substructure temporal logic. In: Logic in computer science'13. IEEE Computer Society, pp 368–377
6. Benerecetti M, Mogavero F, Murano A (2015) Reasoning about substructures and games. Trans Comput Log 16(3):25:1–25:46
7. Berwanger D, Dawar A, Hunter P, Kreutzer S (2006) DAG-width and parity games. In: Symposium on theoretical aspects of computer science'06, LNCS 3884. Springer, New York, pp 524–536
8. Berwanger D, Grädel E (2001) Games and model checking for guarded logics. In: Logic for programming artificial intelligence and reasoning'01, LNCS 2250. Springer, New York, pp 70–84
9. Berwanger D, Grädel E (2004) Fixed-point logics and solitaire games. Theor Comput Sci 37(6):675–694
10. Berwanger D, Grädel E, Kaiser L, Rabinovich R (2012) Entanglement and the complexity of directed graphs. Theor Comput Sci 463:2–25
11. Calude CS, Jain S, Khoussainov B, Li W, Stephan F (2017) Deciding parity games in quasipolynomial time. In: Symposium on theory of computing'17. Association for Computing Machinery, pp 252–263
12. Chatterjee K, Doyen L (2012) Energy parity games. Theor Comput Sci 458:49–60
13. Chatterjee K, Doyen L, Henzinger TA, Raskin J-F (2010) Generalized mean-payoff and energy games. In Foundations of software technology and theoretical computer science'10, LIPIcs 8. Leibniz-Zentrum fuer Informatik, pp 505–516
14. Chatterjee K, Henzinger TA, Horn F (2010) Finitary winning in omega-regular games. Trans Comput Log 11(1):1:1–1:26
15. Chatterjee K, Henzinger TA, Jurdziński M (2005) Mean-payoff parity games. In: Logic in computer science'05. IEEE Computer Society, pp 178–187
16. Chatterjee K, Henzinger TA, Piterman N (2010) Strategy logic. Inf Comput 208(6):677–693
17. Condon A (1992) The complexity of stochastic games. Inf Comput 96(2):203–224
18. Ehrenfeucht A, Mycielski J (1979) Positional strategies for mean payoff games. Int J Game Theory 8(2):109–113
19. Emerson EA, Jutla CS (1991) Tree automata, mu-calculus, and determinacy. In: Foundation of computer science'91. IEEE Computer Society, pp 368–377
20. Emerson EA, Jutla CS, Sistla AP (1993) On model checking for the mu-calculus and its fragments. In: Computer aided verification'93, LNCS 697. Springer, New York, pp 385–396
21. Emerson EA, Jutla CS, Sistla AP (2001) On model checking for the μ-calculus and its fragments. Theor Comput Sci 258(1–2):491–522

22. Emerson EA, Lei C-L (1986) Temporal reasoning under generalized fairness constraints. In: Symposium on theoretical aspects of computer science'86, LNCS 210. Springer, New York, pp 267–278
23. Fearnley J (2010) Non-oblivious strategy improvement. In: Logic for programming artificial intelligence and reasoning'10, LNCS 6355. Springer, New York, pp 212–230
24. Fearnley J, Jain S, Schewe S, Stephan F, Wojtczak D (2017) An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In: SPIN symposium on model checking of software'2017. Association for Computing Machinery, pp 112–121
25. Fearnley J, Lachish O (2011) Parity games on graphs with medium tree-width. In: Mathematical foundations of computer science'11, LNCS 6907. Springer, New York, pp 303–314
26. Fearnley J, Schewe S (2012) Time and parallelizability results for parity games with bounded treewidth. In: International colloquium on automata, languages, and programming'12, LNCS 7392. Springer, pp 189–200
27. Fellows MR, Koblitz N (1992) Self-witnessing polynomial-time complexity and prime factorization. In: Conference on structure in complexity theory'92. IEEE Computer Society, pp 107–110
28. Fellows MR, Koblitz N (1992) Self-witnessing polynomial-time complexity and prime factorization. Des Codes Crypt 2(3):231–235
29. Fijalkow N, Zimmermann M (2012) Cost-parity and cost-streett games. In: Foundations of software technology and theoretical computer science'12, LIPIcs 18. Leibniz-Zentrum fuer Informatik, pp 124–135
30. Fijalkow N, Zimmermann M (2014) Cost-parity and cost-streett games. Log Methods Comput Sci 10(2):1–29
31. Friedmann O, Lange M (2009) Solving parity games in practice. In: Automated technology for verification and analysis'09, LNCS 5799. Springer, pp 182–196
32. Grädel E, Thomas W, Wilke T (2002) Automata, logics, and infinite games: a guide to current research. LNCS 2500. Springer, New York
33. Gurvich VA, Karzanov AV, Khachivan LG (1990) Cyclic games and an algorithm to find minimax cycle means in directed graphs. USSR Comput Math Math Phys 28(5):85–91
34. Jurdziński M (1998) Deciding the winner in parity games is in UP ∩ co-UP. Inf Process Lett 68(3):119–124
35. Jurdziński M (2000) Small progress measures for solving parity games. In: Symposium on theoretical aspects of computer science'00, LNCS 1770. Springer, pp 290–301
36. Jurdziński M, Lazic R (2017) Succinct progress measures for solving parity games. In: Logic in computer science'17. Association for Computing Machinery. Accepted for publication, pp 1–9
37. Jurdziński M, Paterson M, Zwick U (2006) A deterministic subexponential algorithm for solving parity games. In: Symposium on discrete algorithms'06. Society for Industrial and Applied Mathematics, pp 117–123
38. Jurdziński M, Paterson M, Zwick U (2008) A Deterministic Subexponential Algorithm for Solving Parity Games. SIAM J Comput 38(4):1519–1532
39. Klarlund N, Kozen D (1991) Rabin measures and their applications to fairness and automata theory. In: Logic in computer science'91. IEEE Computer Society, pp 256–265
40. Kupferman O, Vardi MY (1998) Weak alternating automata and tree automata emptiness. In: Symposium on theory of computing'98. Association for Computing Machinery, pp 224–233
41. Martin AD (1975) Borel determinacy. Ann Math 102(2):363–371
42. Martin AD (1985) A purely inductive proof of Borel determinacy. In: Symposia in pure mathematics'82, recursion theory. American Mathematical Society and Association for Symbolic Logic, pp 303–308
43. McNaughton R (1993) Infinite games played on finite graphs. Ann Pure Appl Log 65:149–184
44. Mogavero F, Murano A, Perelli G, Vardi MY (2012) What makes ATL* decidable? A decidable fragment of strategy logic. In: Concurrency theory'12, LNCS 7454. Springer, Berlin, pp 193–208
45. Mogavero F, Murano A, Perelli G, Vardi MY (2014) Reasoning about strategies: on the model-checking problem. Trans Comput Log 15(4):341–3442
46. Mogavero F, Murano A, Sorrentino L (2013) On promptness in parity games. In: Logic for programming artificial intelligence and reasoning'13, LNCS 8312. Springer, New York, pp 601–618
47. Mogavero F, Murano A, Vardi MY (2010) Reasoning about strategies. In: Foundations of software technology and theoretical computer science'10, LIPIcs 8. Leibniz-Zentrum fuer Informatik, pp 133–144
48. Mostowski AW (1984) Regular expressions for infinite trees and a standard form of automata. In: Symposium on computation theory'84, LNCS 208. Springer, New York, pp 157–168
49. Mostowski AW (1991) Games with forbidden positions. Technical report, University of Gdańsk, Gdańsk, Poland
50. Obdrzálek J (2003) Fast mu-calculus model checking when tree-width is bounded. In: Computer aided verification'03, LNCS 2725. Springer, New York, pp 80–92

51. Obdrzálek J (2007) Clique-width and parity games. In: Computer science logic'07, LNCS 4646. Springer, New York, pp 54–68
52. Schewe S (2007) Solving parity games in big steps. In: Foundations of software technology and theoretical computer science'07, LNCS 4855. Springer, New York, pp 449–460
53. Schewe S (2008) An optimal strategy improvement algorithm for solving parity and payoff games. In: Computer science logic'08, LNCS 5213. Springer, New York, pp 369–384
54. Schewe S (2008) ATL* satisfiability is 2EXPTIME-complete. In: International colloquium on automata, languages, and programming'08, LNCS 5126. Springer, New York, pp 373–385
55. Schewe S, Finkbeiner B (2006) Satisfiability and finite model property for the alternating-time μ-calculus. In: Computer science logic'06, LNCS 6247. Springer, New York, pp 591–605
56. Vöge J, Jurdziński M (2000) A discrete strategy improvement algorithm for solving parity games. In: Computer aided verification'00, LNCS 1855. Springer, New York, pp 202–215
57. Wilke T (2001) Alternating tree automata, parity games, and modal μ-calculus. Bull Belg Math Soc 8(2):359–391
58. Zielonka W (1998) Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor Comput Sci 200(1–2):135–183
59. Zwick U, Paterson M (1996) The complexity of mean payoff games on graphs. Theor Comput Sci 158(1–2):343–359