Journal of Computer and System Sciences 147 (2025) 103580

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

journal homepage: www.elsevier.com/locate/jcss

Priority Promotion with Parysian flair

Massimo Benerecetti [a], Daniele Dell'Erba [b,*], Fabio Mogavero [a], Sven Schewe [b], Dominik Wojtczak [b]

[a] Università degli Studi di Napoli Federico II, Naples, Italy
[b] University of Liverpool, Liverpool, UK

ARTICLE INFO

Article history:
Received 3 April 2023
Received in revised form 24 July 2024
Accepted 7 August 2024
Available online 28 August 2024

Keywords:
Parity games
Quasi dominion
Quasi polynomial algorithms

ABSTRACT

We develop an algorithm that combines the advantages of Priority Promotion, that is one of the leading approaches to solving large parity games in practice, with the quasi-polynomial time guarantees offered by Parys' algorithm. Hybridising these algorithms sounds both natural and difficult, as they both generalise the classic recursive algorithm in different ways that appear to be irreconcilable: while the promotion transcends the call structure, the guarantees change on each level. We show that an interface that respects both is not only effective, but also efficient.

© 2024 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Parity games are perfect-information two-player turn-based games of infinite duration, usually played on finite directed graphs. Their vertices, labelled by natural numbers, called *priorities*, are assigned to one of two players, named *Even* and *Odd* (or simply 0 and 1, respectively). A play in the game consists of a sequence of moves between vertices and it is said to be winning for player 0 (*resp.*, 1), if the maximal priority encountered infinitely often along the play is even (*resp.*, odd). Parity games have many applications in the context of formal verification and synthesis of systems. Computing winning strategies for these games is linear-time equivalent to solving the modal μ-calculus model-checking problem [1,2]. Parity games have been studied in automata theory [3–5] and can be applied to solve the complementation problem for alternating automata [6] or the emptiness of the corresponding nondeterministic tree automata [7]. These automata, in turn, can be used to solve the satisfiability and model-checking problems for several expressive logics [8–14], such as μ-calculus [15,16] and ATL* [17,18]. On the complexity-theoretic side, determining the winner of a parity game is a problem that lies in NP ∩ co-NP [1], being memoryless determined [19–21]. It has even been proved to belong to UP ∩ co-UP [22], and later to be solvable in quasi-polynomial time [23]. However, determining whether they belong also to P is still an open challenge.

Research on parity games falls into two categories: on the one hand to develop *fast solvers*; on the other hand to determine the *complexity of parity games* by finding algorithms with a good *worst-case complexity*. With its practical motivation, one of the most efficient approach for solving parity games is currently the *Priority Promotion* technique [24–28], a derivation of the classic *recursive algorithm* [29,30] that follows the iterated fixed-point structure induced by the parity condition. Like all of practically efficient algorithms, it is an exponential approach, however, algorithms that are good in practice do not tend to display their worst-case behaviour, except in carefully designed hostile examples. This holds in particular for strategy improvement algorithms [31–37], which were considered candidates for tractable algorithms until they were shown

* Corresponding author.
E-mail addresses: massimo.benerecetti@unina.it (M. Benerecetti), dde@liverpool.ac.uk (D. Dell'Erba), fabio.mogavero@unina.it (F. Mogavero), svens@liverpool.ac.uk (S. Schewe), dkw@liverpool.ac.uk (D. Wojtczak).

https://doi.org/10.1016/j.jcss.2024.103580
0022-0000/© 2024 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

to be exponential by Friedman's delicate lower bound constructions [38–40] and by [41] for the symmetric approach of [37]. But while it is easier to design hard classes for recursive [42–44] and Priority Promotion algorithms [28], these classes are still not relevant in practice.

One of the most celebrated results in recent years has been the landmark result of [23], which established that parity games can be solved in quasi-polynomial time (QP). This was a major step from former deterministic algorithms, which were (at least) exponential in the number of priorities [29,2,45,46,30,47,34,35,48,27] ($n^{O(c)}$), or in the square-root of the number of game positions [31,49,34] (approximately $n^{O(\sqrt{n})}$). The breakthrough of [23] has triggered a new line of research into QP algorithms, including its refinements [50–52], the register-index algorithm [53,54], the study on the Strahler Number [55], and the bounded version of the recursive algorithm [56]. Unfortunately, all these quasi-polynomial algorithms can be derived by the separation approach that also provides a lower bound for these techniques [57].

Interestingly, Parys' algorithm [56] and variations thereof [58,59], which, like Priority Promotion techniques, adjust the classic recursive algorithm [29,30], are relatively fast among the QP algorithms, where [56] has the edge on benchmarks, while [58] has the edge on theoretical guarantees. On first glance, this seems to invite synthesising one of these algorithms with Priority Promotion. This upgrade would combine the best of both worlds: offer QP lower bounds without undue compromise on efficiency. On second glance, the prospect of this synthesis seems less promising. Priority Promotion techniques [26–28,60–62] achieve their advancement over the classic recursive algorithm [30] by globally bypassing the call structure through temporally increasing the priority of a position. Parys' approach, on the other hand, locally creates sets with guarantees weakened (halved) along the recursive call structure, where subgames are split into areas that contain all 0-dominions with size up to a bound b_0 and all 1-dominions of size up to a bound b_1; one of these bounds is halved in each call until the guarantees are trivial. *Prima facie*, it seems clear that such guarantees are ill suited for a promotion across the call structure. We did, however, find that, when one shifts the view on the essence of a promotion from creating quasi-dominions to creating regions and promoting them to the lowest level where they are no longer dominions, this allows for a concurrent treatment of sets with bounded guarantees (the Parysian flair of our hybrid algorithm) and with unbounded guarantees (the Priority Promotion core of our algorithm). While the integration of these seemingly antagonistic concepts is intricate, it provides an efficient bridge between the behaviour and the data structures of [27] and [56]: the resulting algorithm guarantees a quasi-polynomial running time, and offers excellent practical behaviour on the benchmarks we have tested it against.

2. Preliminaries

A two-player turn-based *arena* is a tuple $\mathcal{A} = \langle \mathrm{Ps}_0, \mathrm{Ps}_1, \mathsf{Mv} \rangle$, with $\mathrm{Ps}_0 \cap \mathrm{Ps}_1 = \emptyset$ and $\mathrm{Ps} \triangleq \mathrm{Ps}_0 \cup \mathrm{Ps}_1$, such that $\langle \mathrm{Ps}, \mathsf{Mv} \rangle$ is a finite directed graph without sinks. Ps_0 (*resp.*, Ps_1) is the set of positions of the Even (*resp.*, Odd) and $\mathsf{Mv} \subseteq \mathrm{Ps} \times \mathrm{Ps}$ 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.*, $(\pi_i, \pi_{i+1}) \in \mathsf{Mv}$, for all $i \in [0, |\pi| - 1)$. A positional *strategy* for player $\alpha \in \{0, 1\}$ on $\mathrm{V} \subseteq \mathrm{Ps}$ is a function $\sigma_\alpha \in \mathrm{Str}_\alpha(\mathrm{V}) \subseteq (\mathrm{V} \cap \mathrm{Ps}_\alpha) \to \mathrm{V}$, mapping each α-position v in V to position $\sigma_\alpha(v)$ compatible with the move relation, *i.e.*, $(v, \sigma_\alpha(v)) \in \mathsf{Mv}$. With $\mathrm{Str}_\alpha(\mathrm{V})$ we denote the set of all α-strategies on V. When talking about players, we will refer to the opponent player of α as $\overline{\alpha}$. 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 $\mathsf{play}((\sigma_0, \sigma_1), v)$. Given a partial function $\mathsf{f} : \mathrm{A} \rightharpoonup \mathrm{B}$, with $\mathsf{dom}(\mathsf{f}) \subseteq \mathrm{A}$ and $\mathsf{rng}(\mathsf{f}) \subseteq \mathrm{B}$ we denote the domain and the range of f, respectively.

A *parity game* is a tuple $\mathcal{G} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathsf{PG}$, 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: $\mathsf{pr}(\mathcal{G}) \triangleq \max_{v \in \mathrm{Ps}} \mathsf{pr}(v)$; for a path $\pi \in \mathrm{Pth}$, we define $\mathsf{pr}(\pi) \triangleq \max_{i \in [0, |\pi|)} \mathsf{pr}((\pi)_i)$, 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. The *winning region* for player $\alpha \in \{0, 1\}$ in game $\mathcal{G}$, denoted by $\mathrm{Wn}^\alpha_\mathcal{G}$, is the greatest set of positions that is also a α-dominion in $\mathcal{G}$. Since parity games are memoryless determined [20], meaning that from each position one of the two players wins, the two winning regions of a game $\mathcal{G}$ form a partition of its positions, *i.e.*, $\mathrm{Wn}^0_\mathcal{G} \cup \mathrm{Wn}^1_\mathcal{G} = \mathrm{Ps}_\mathcal{G}$.

The (maximal) subgame $\mathcal{G}' = \mathcal{G} \setminus \mathrm{V}$ of $\mathcal{G}$ is the game 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 $\mathsf{pre}^\alpha(\mathrm{V}) \triangleq \{v \in \mathrm{Ps}_\alpha \mid \mathsf{Mv}(v) \cap \mathrm{V} \neq \emptyset\} \cup \{v \in \mathrm{Ps}_{\overline{\alpha}} \mid \mathsf{Mv}(v) \subseteq \mathrm{V}\}$, collects the positions from which player α can force the game to reach some position in V with a single move. The α-attractor $\mathsf{atr}^\alpha(\mathrm{V})$ generalises the notion of α-predecessor $\mathsf{pre}^\alpha(\mathrm{V}) \cup \mathrm{V}$ to an arbitrary number of moves. Thus, it corresponds to the least fix-point of that operator. When $\mathrm{V} = \mathsf{atr}^\alpha(\mathrm{V})$, player α cannot force any position outside V to enter this set that is, therefore, called α-maximal. For such a V, the set of positions of the subgame $\mathcal{G} \setminus \mathrm{V}$ is precisely $\mathrm{Ps} \setminus \mathrm{V}$. When the computation of the attractor is restricted to a given set of positions X, we will use the notation $\mathsf{atr}^\alpha(\mathrm{V}, \mathrm{X})$ which corresponds to the least fix-point of $\mathsf{pre}^\alpha(\mathrm{V}) \cap \mathrm{X}$. 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. Observe that all the operators and sets described above actually depend on the specific game $\mathcal{G}$ they are applied to. In the rest of the paper, we shall only add $\mathcal{G}$ as subscript of an operator, *e.g.* $\mathsf{atr}^\alpha_\mathcal{G}(\mathrm{V})$, when the game is not clear from the context.

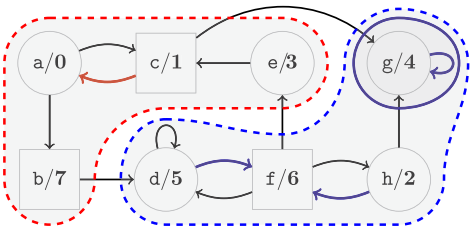

Fig. 1. A game and some of its quasi dominions. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

Outline

We introduce our hybrid algorithm in three steps. In the first step (Section 3), we introduce a variation of the classic Priority Promotion approach, which serves as the backbone of our hybrid algorithm in Section 5. We provide a recap of how Priority Promotion operates and an introduction to its data structure, which we later extend for our hybrid algorithm. In a nutshell, Priority Promotion accelerates the classic recursive algorithm by allowing to merge dominions in subgames that span non-adjacent recursive calls, which is the essence of the promotion operations. In the subsequent section (Section 4), we outline Parys' algorithm, which does not seek to identify *all dominions* on a level, but merely *all small dominions* up to given bounds b_0 and b_1 for the dominions of Player 0 and Player 1, respectively. It truncates the size of the call tree by making all but one call with half the precision for one of the players. Here, we formulate the algorithm with a terminology analogous to Priority Promotion, and present it in a form similar to our hybrid algorithm.

The two concepts of Priority Promotion and truncated tree size through limited guarantees appear to be unlikely allies: not only does the presence of Parys' sets with limited guarantees impede the promotion of dominions, any attempt to promote sets with bounded guarantees is doomed to fail, when the bounds are larger (and thus the required guarantees stronger) along the call tree. In Section 5, we see that, when synthesising the algorithms carefully, sets with the 'region guarantees' from Priority Promotion and with 'bounded guarantees' from Parys' approach can co-exist, so long as they are kept carefully apart and treated differently.

The resulting algorithm can identify dominions in many places, and these dominions can be promoted. This promotion can be to a set with 'region guarantees' at a higher level, but it can also be that the correct target is a set with 'bounded guarantees'; this works across levels because dominions have unbounded guarantees. The identification of the right set to promote to, instead, remains fairly similar to the way such a set is identified in classic Priority Promotion. While sets with bounded guarantees cannot be promoted along the data structure (which follows the call tree), they lose parts of their locality: positions can be promoted into them, and, crucially, they do not prevent promotions to higher levels. This way, we can keep the Priority Promotion part (which usually carries the main burden of solving the parity game) and can play out its practical efficiency in full, while we also retain the quasi-polynomial complexity from Parys' algorithm [56], bypassing the known hard cases for recursive algorithms. For practical considerations, it is still computationally attractive to grow the bounded sets more slowly: we found that some of the points where Parys' algorithm applies a closure of sets with bounded guarantees are merely for the convenience of the proof. For efficiency, we have restricted the closure under attractor of these sets to the places where it is necessary for correctness.

3. The Priority Promotion approach

The *Priority Promotion approaches* [24–28] attack the problem of solving a parity game $\mathcal{D} \in \mathsf{PG}$ by iteratively computing, one at a time, a sequence of α-dominions $\mathrm{D}_0^\alpha, \mathrm{D}_1^\alpha, \ldots \subseteq \mathrm{Ps}$, for some player $\alpha \in \mathbb{B} \triangleq \{0, 1\}$ (potentially, for both of them). These are in fact portions of the two winning regions, Wn_0 and Wn_1, that need to be identified (the original notion of dominion has been introduced in [63,49]). The idea here is to start from a weaker notion, called *quasi dominion*. Similarly to a dominion, a quasi dominion is a set of positions over which one of the two players has a strategy defined on that set, whose induced plays, if infinite, are winning for that player. In contrast to dominions, it's worth noting that some of these plays could be finite. This is because the opponent may have the possibility to escape from those positions towards a different part of the game, hoping for a better outcome. The approach uses quasi dominions by composing them until an actual dominion is obtained. The name of the approach refers to the principle that this composition is computed by applying the following operation called *promotion*. Given two quasi dominions Q_1 and Q_2, to which some priorities $p_1 < p_2$ of the same parity are assigned, Q_1 is combined with Q_2 by promoting the former to the priority of the latter. As a result, together they form a bigger quasi dominion $\mathrm{Q}_1 \cup \mathrm{Q}_2$, whose internal infinite plays are still winning for the same player.

Definition 1 *(Quasi dominion).* A set of positions $\mathrm{Q} \subseteq \mathrm{Ps}$ is a *quasi α-dominion*, for some player $\alpha \in \mathbb{B}$, if there exists an α-strategy $\sigma_\alpha \in \mathrm{Str}^\alpha(\mathrm{Q})$, called *$\alpha$-witness for* Q, such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}^{\overline{\alpha}}(\mathrm{Q})$ and positions $v \in \mathrm{Q}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$, satisfies $\mathsf{pr}(\pi) \equiv_2 \alpha$, if infinite.

The usefulness of the above concept, in addition to the property of being suitably composable, resides in the fact that quasi dominions are closed under inclusion. Thus, when a subset of an α-quasi dominion is found for which player $\overline{\alpha}$ has no strategy that leaves the set, a dominion is identified. Note that here no constraints are imposed on the finite plays, unlike in the definition given in the Priority Promotion paper [24], where quasi dominions were not closed under inclusion. However, an analogous constraint is implicitly imposed on the basic data structure leveraged by the algorithm and introduced in Definition 4 below. It is interesting to notice that this version of quasi dominion is similar to the notion of a *snare* [36].

Theorem 1 *(Induced dominion). Let $\alpha \in \mathbb{B}$ be a player, $\mathrm{Q} \subseteq \mathrm{Ps}$ a quasi α-dominion, $\sigma \in \mathrm{Str}_\alpha(\mathrm{Q})$ one of its α-witnesses, and $\mathrm{D} \subseteq \mathrm{Q}$ a subset such that $\sigma(v) \in \mathrm{D}$, if $v \in \mathrm{Ps}_\alpha$, and $\mathsf{Mv}(v) \subseteq \mathrm{D}$, otherwise, for all positions $v \in \mathrm{D}$. Then, D is an α-dominion.*

Example 1. As an example of quasi dominions, consider the game depicted in Fig. 1, where circle-shaped positions belong to Player `0` and square-shaped ones to Player `1`. Clearly, `g` and `h` are won by Player `0`, while the rest of the game is won by Player `1`. The dashed blue region, containing the four positions `d`, `f`, `g`, and `h`, is one of the quasi `0`-dominion in the game, where one of its two possible `0`-witnesses is highlighted in blue (the other one being the `0`-strategy where Player `0` moves from `h` to `g`). The solid blue subregion is a `0`-dominion induced by the single position `g`. The dashed red region, containing the remaining half of the game, is instead a quasi `1`-dominion with the `1`-witnesses highlighted in red. Finally, note that this region is also a quasi `0`-dominion with `0`-witness mapping position `a` to `b`. Indeed, any play compatible with this witness will eventually exit the region. In general, a set of position is both a quasi `0`-dominion and a quasi `1`-dominion if both players can force exiting the set.

The solution algorithms following this approach carry on the search for a dominion by exploring a *finite strict partial order* $\langle \mathrm{St}, s_I, \prec \rangle$. Its elements, called *states*, record information about the quasi dominions computed up to that point. In the *initial state* s_I, the quasi dominions are initialised to the sets of positions with the same priority. At each step, a new quasi α-dominion Q, for some player $\alpha \in \mathbb{B}$, is *extracted* from the current state s and used to compute a *successor state w.r.t.* the order $\prec$, if Q is *open*, *i.e.*, if it is not an α-dominion. If, on the other hand, it is *closed*, then the search is over and Q is added to the portion of the winning region Wn_α computed so far.

We describe here a version of the Priority Promotion algorithm that instantiates the above partial order and serves as a basis for the hybrid approach presented later in this section. To do so, we shall need some technical notions, all of which refer to some fixed parity game $\mathcal{D} \in \mathsf{PG}$. By $\mathrm{Pr}_\bot \triangleq \mathrm{Pr} \cup \{\bot\}$ and $\mathrm{Pr}_\top \triangleq \mathrm{Pr} \cup \{\top_0, \top_1\}$ we denote the set of priorities in $\mathcal{D}$ extended with the bottom symbol $\bot$ and two top symbols $\top_0$ and $\top_1$, one for each player. The symbols $\bot$, $\top_0$, and $\top_1$ are called *pseudo priorities*. The standard ordering $<$ on Pr is extended to these additional elements in the natural way: $\bot$ is the smallest element, while both $\top_0$ and $\top_1$ are strictly greater than every other priority; we do not assume any specific order between the two maximal elements, though, we consider $\top_0$ even and $\top_1$ odd.

The first step in the formalisation of the notion of state requires the concept of *priority-lift function*, which is at the basis of all the approaches discussed in this article. Intuitively, it is a partial function from positions to priorities that over-approximates the priority function of the game.

Definition 2 *(Priority-lift function).* A *priority-lift function* $\mathsf{r} \in \mathrm{Pf} \triangleq \mathrm{Ps} \rightharpoonup \mathrm{Pr}_\top$ is a partial function such that $\mathsf{r}(v) \geq \mathsf{pr}(v)$, for every position $v \in \mathsf{dom}(\mathsf{r})$.

In the following, we adopt the same notation as in [27]. Given a priority-lift function $\mathsf{r} \in \mathrm{Pf}$ and a priority $p \in \mathrm{Pr}$, we denote with $\mathsf{r}^{(\sim p)}$, for $\sim \in \{<, \leq, \geq, >\}$, the function obtained by restricting the domain of r to the positions $v \in \mathsf{dom}(\mathsf{r})$ whose priority $\mathsf{r}(v)$ satisfies the relation $\mathsf{r}(v) \sim p$, *i.e.*, $\mathsf{r}^{(\sim p)} \triangleq \mathsf{r} \restriction \{v \in \mathsf{dom}(\mathsf{r}) \mid \mathsf{r}(v) \sim p\}$, where $\restriction$ is the standard operation of domain restriction. By $\Delta_\mathsf{r}^\alpha \triangleq \{v \in \mathsf{dom}(\mathsf{r}) \mid \mathsf{r}(v) \equiv_2 \alpha\}$ we denote the set of positions in r with a priority congruent to $\alpha \in \mathbb{B}$, and with $\Delta_\mathsf{r}^{\alpha,p} \triangleq \left\{v \in \Delta_\mathsf{r}^\alpha \,\middle|\, \mathsf{r}(v) \geq p\right\}$ its subset with priorities greater than or equal to p.

Example 2. Consider again the game reported in Fig. 1 and the total function $\mathsf{r} \triangleq \{\mathtt{a}, \mathtt{c}, \mathtt{e} \mapsto 3; \mathtt{b} \mapsto 7; \mathtt{d}, \mathtt{f}, \mathtt{h} \mapsto 6; \mathtt{g} \mapsto \top_0\}$ from positions to priorities. It is immediate to see that r is a priority-lift function such that Δ_r^0 and Δ_r^1 denote the dashed blue and red regions, respectively. Moreover, $\Delta_\mathsf{r}^{0,7} = \{\mathtt{g}\}$, while $\Delta_\mathsf{r}^{1,7} = \{\mathtt{b}\}$.

A state encodes information about the quasi dominions computed up to a certain point of the computation. To this end, we require that all the positions in a priority-lift function r with priority of parity $\alpha \in \mathbb{B}$, *i.e.*, the set Δ_r^α, form a quasi α-dominion. Moreover, the idea is to store all α-dominions already identified by associating them with the corresponding pseudo priority $\top_\alpha$.

Definition 3 *(Quasi-dominion function).* A *quasi-dominion function* $\mathsf{r} \in \mathrm{Qs} \subseteq \mathrm{Pf}$ is a priority-lift function satisfying the following conditions, for every $\alpha \in \mathbb{B}$:

1. the set Δ_r^α is a quasi α-dominion;
2. the set $\mathsf{r}^{-1}(\top_\alpha)$ is an α-dominion.

The notion of quasi-dominion function represents the backbone of the algorithm, being the data structure to which the promotion operation is applied.

Example 3. It is immediate to see that the priority-lift function r reported in Example 2 is actually a quasi-dominion function. Indeed, as observed in Example 1, the two sets $\Delta_{\mathsf{r}}^0 = \{\mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and $\Delta_{\mathsf{r}}^1 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e}\}$ are quasi dominions for Player $\mathtt{0}$ and $\mathtt{1}$, respectively; in addition, $\mathsf{r}^{-1}(\top_{\mathtt{0}}) = \{\mathtt{g}\}$ is a $\mathtt{0}$-dominion. Since, by definition, an empty set is always a dominion, $\mathsf{r}^{-1}(\top_{\mathtt{1}}) = \emptyset$ is a $\mathtt{1}$-dominion too, which concludes the verification of the properties required by Definition 3.

An important property of dominions is that the extension by means of its α-attractor, namely the set of positions obtained by applying the α-attractor operator, of an α-dominion Q is still an α-dominion. This property, however, is not enjoyed by arbitrary quasi dominions. Indeed, there may even be cases where the α-attractor of a quasi α-dominion is a $\overline{\alpha}$-dominion. Moreover, to efficiently verify whether a quasi dominion is actually a dominion, an explicit representation of one of its witnesses is usually required. To overcome these complications, we consider a well-behaving subclass of quasi dominions that meets the following requirements:

1. the set $\mathsf{esc}(\mathsf{Q}, \sigma) \triangleq \{v \in \mathsf{Ps}_\alpha \cap \mathsf{Q} \mid \sigma(v) \notin \mathsf{Q}\}$ of α-positions, which leave a quasi α-dominion $\mathsf{Q} \subseteq \mathsf{Ps}$ by following one of its α-witnesses $\sigma \in \mathsf{Str}^\alpha(\mathsf{Q})$, is a subset of the $\overline{\alpha}$-escape positions $\mathsf{esc}^{\overline{\alpha}}(\mathsf{Q})$;
2. all these $\overline{\alpha}$-escape positions have priorities congruent to α and greater than the ones of the positions that can be attracted to Q by Player α.

The first requirement states the strategy σ of player α is allowed to leave Q only if it is forced to do so. This ensures that, in order to verify whether Q is an α-dominion, it suffices to check for the emptiness of $\mathsf{esc}^{\overline{\alpha}}(\mathsf{Q})$ (see [26, Definition 1]). The second one, instead, ensures closure under extension by α-attractor. Indeed, since only positions with lower priorities than those of $\mathsf{esc}^{\overline{\alpha}}(\mathsf{Q})$ can be attracted by Q, every cycle induced by the attractor strategy inside Q is necessarily winning for player α (see [24, Proposition 2]). As observed before, a similar structure, which retains weaker properties, is known as snare [36].

Definition 4 *(Region function).* A *region function* $\mathsf{r} \in \mathsf{Rg} \subseteq \mathsf{Qs}$ is a quasi-dominion function satisfying the following conditions, for every $\alpha \in \mathbb{B}$:

1. there exists an α-witness $\sigma_\alpha \in \mathsf{Str}^\alpha(\Delta_{\mathsf{r}}^\alpha)$ for the quasi α-dominion $\Delta_{\mathsf{r}}^\alpha$ such that $\mathsf{esc}(\Delta_{\mathsf{r}}^{\alpha,p}, \sigma_\alpha) \subseteq \mathsf{esc}^{\overline{\alpha}}(\Delta_{\mathsf{r}}^{\alpha,p})$, for all priorities $p \in \mathsf{rng}(\mathsf{r})$, with $p \equiv_2 \alpha$;
2. for all priorities $p \in \mathsf{rng}(\mathsf{r})$, with $p \equiv_2 \alpha$, and positions $v \in \mathsf{esc}^{\overline{\alpha}}(\Delta_{\mathsf{r}}^{\alpha,p})$, it holds that $p \leq \mathsf{pr}(v) \equiv_2 \alpha$.

Example 4. As observed in Example 3, the priority-lift function r of Example 2 is a quasi-dominion function. We can now show that it is a region function as well. Indeed, we have (1) $\Delta_{\mathsf{r}}^{1,3} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{e}\}$, (2) $\Delta_{\mathsf{r}}^{0,6} = \{\mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$, and (3) $\Delta_{\mathsf{r}}^{1,7} = \{\mathtt{b}\}$; moreover, (1) $\mathsf{esc}(\Delta_{\mathsf{r}}^{1,3}, \sigma_1) = \mathsf{esc}^0(\Delta_{\mathsf{r}}^{1,3}) = \mathsf{esc}(\Delta_{\mathsf{r}}^{1,7}, \sigma_1) = \mathsf{esc}^0(\Delta_{\mathsf{r}}^{1,7}) = \{\mathtt{b}\}$ and (2) $\emptyset = \mathsf{esc}(\Delta_{\mathsf{r}}^{0,6}, \sigma_0) \subseteq \mathsf{esc}^1(\Delta_{\mathsf{r}}^{0,6}) = \{\mathtt{f}\}$, where σ_0 and σ_1 are the witnesses of the two quasi dominions highlighted in Fig. 1 in blue and red, respectively; moreover, $\mathsf{pr}(\mathtt{b}) = 7 \equiv_2 \mathtt{1}$, which is greater than or equal to the odd priorities in r, namely 3 and 7. Another example of region function is $\mathsf{r} = \{\mathtt{a} \mapsto 0; \mathtt{c} \mapsto 1; \mathtt{e} \mapsto 3; \mathtt{d}, \mathtt{f}, \mathtt{h} \mapsto 6; \mathtt{b} \mapsto 7; \mathtt{g} \mapsto \top_{\mathtt{0}}\}$, which is displayed on the left-hand side of Fig. 2, and where $\Delta_{\mathsf{r}}^{0,0} = \{\mathtt{a}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$, $\Delta_{\mathsf{r}}^{1,1} = \{\mathtt{b}, \mathtt{c}, \mathtt{e}\}$, $\Delta_{\mathsf{r}}^{1,3} = \{\mathtt{b}, \mathtt{e}\}$, $\Delta_{\mathsf{r}}^{0,6} = \{\mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$, and $\Delta_{\mathsf{r}}^{1,7} = \{\mathtt{b}\}$. The easy verification of the properties stated in Definitions 2 and 3 is left to the reader, so we shall only focus on Definition 4. By computing the escape sets for the five priorities $0, 1, 3, 6, 7 \in \mathsf{rng}(\mathsf{r})$, we obtain the following, where σ_0 and σ_1 are two arbitrary witnesses compatible with the partial ones highlighted in the figure: (1) $\{\mathtt{a}\} = \mathsf{esc}(\Delta_{\mathsf{r}}^{0,0}, \sigma_0) \subseteq \mathsf{esc}^1(\Delta_{\mathsf{r}}^{0,0}) = \{\mathtt{a}, \mathtt{f}\}$; (2) $\mathsf{esc}(\Delta_{\mathsf{r}}^{1,1}, \sigma_1) = \mathsf{esc}^0(\Delta_{\mathsf{r}}^{1,1}) = \{\mathtt{b}, \mathtt{c}\}$; (3) $\{\mathtt{b}\} = \mathsf{esc}(\Delta_{\mathsf{r}}^{1,3}, \sigma_1) \subseteq \mathsf{esc}^0(\Delta_{\mathsf{r}}^{1,3}) = \{\mathtt{b}, \mathtt{e}\}$; (4) $\emptyset = \mathsf{esc}(\Delta_{\mathsf{r}}^{0,6}, \sigma_0) \subseteq \mathsf{esc}^1(\Delta_{\mathsf{r}}^{0,6}) = \{\mathtt{f}\}$; (5) $\mathsf{esc}(\Delta_{\mathsf{r}}^{1,7}, \sigma_1) = \mathsf{esc}^0(\Delta_{\mathsf{r}}^{1,7}) = \{\mathtt{b}\}$. The first requirement of Definitions 4 is, thus, satisfied. The verification of the second requirement is immediate, once we observe that (1) $\mathsf{pr}(\mathtt{a}), \mathsf{pr}(\mathtt{f}) \geq 0$, (2) $\mathsf{pr}(\mathtt{b}), \mathsf{pr}(\mathtt{c}), \mathsf{pr}(\mathtt{e}) \geq 1$, (3) $\mathsf{pr}(\mathtt{b}), \mathsf{pr}(\mathtt{e}) \geq 3$, (4) $\mathsf{pr}(\mathtt{f}) \geq 6$, and (5) $\mathsf{pr}(\mathtt{b}) \geq 7$.

Note that every set $\Delta_{\mathsf{r}}^{\alpha,p}$, with $p \in \mathsf{rng}(\mathsf{r})$, is a quasi α-dominion, being a subset of the quasi α-dominion $\Delta_{\mathsf{r}}^\alpha$. Also, it is immediate to see that the priority function pr of a given parity game $\mathcal{D}$ is always a region function. Indeed, it is trivially a priority-lift function. Moreover, the positions in $\Delta_{\mathsf{pr}}^\alpha$, namely those with priorities of parity α, form a quasi α-dominion, whose α-witness can be any strategy that chooses to remain inside the set whenever this is allowed by the move relation. Thus, it is a quasi-dominion function as well. Finally, since $\Delta_{\mathsf{pr}}^{\alpha,p}$ cannot contain positions of parity $\overline{\alpha}$ and thanks to the way the α-witness is chosen, it is clear that pr also satisfies the conditions of Definition 4.

At this point, we have the technical tools to introduce the *search space* that instantiates the (finite) strict partial order mentioned in the intuitive explanation of the approach given above. In particular, to account for the current status of the search of a dominion in a game $\mathcal{D}$, we define a *state* s as a pair (r, p), comprising a region function r and a priority p, with the idea that

1. all quasi α-dominions computed so far are contained in $\Delta_{\mathsf{r}}^{\alpha,q}$, for some priority q strictly greater than p,

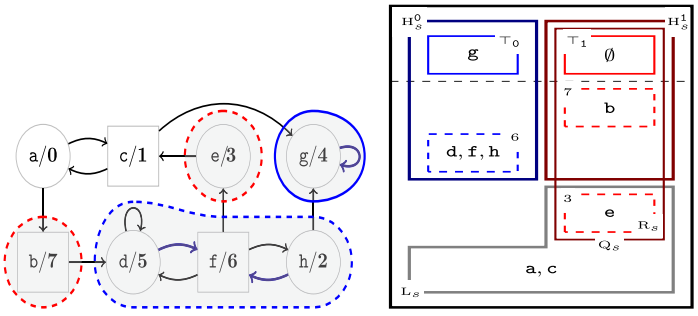

Fig. 2. A game and a corresponding state representation.

2. the current quasi dominion to focus on is contained in r at priority p, and
3. all positions with priorities smaller than or equal to p correspond to the portion of the game that still has to be processed.

Given a state $s = (\mathsf{r}, p)$, we denote by $\mathsf{r}_s \triangleq \mathsf{r}$ and $p_s \triangleq p$ its two components and by $\alpha_s \triangleq p_s \bmod 2$ the parity of the state, *i.e.*, the player associated to the next quasi dominion to process. Moreover, we set (i) $\mathrm{Q}_s \triangleq \Delta^{\alpha_s, p_s}_{\mathsf{r}_s}$, (ii) $\mathrm{R}_s \triangleq \mathsf{r}_s^{-1}(p_s)$, (iii) $\mathrm{L}_s \triangleq \{v \in \mathsf{dom}(\mathsf{r}_s) \mid \mathsf{r}_s(v) \leq p_s\}$, and (iv) $\mathrm{H}^\alpha_s \triangleq \Delta^\alpha_{\mathsf{r}_s} \setminus \mathrm{L}_s$, for all $\alpha \in \mathbb{B}$. We refer to Q_s as *current quasi dominion* and to L_s as the *local area*, *i.e.*, the set of unprocessed positions yet to be analysed (see Item 3 above). Both sets also include the quasi α_s-dominion R_s on which the next step of the search will focus (see Item 2 above). More precisely, R_s, called *α_s-region* or simply *region* whenever the player is clear from the context, is the intersection of Q_s and L_s. As observed above, the two quasi dominions $\Delta^0_{\mathsf{r}_s}$ and $\Delta^1_{\mathsf{r}_s}$ partition the entire set of positions in the game, thus, H^0_s and H^1_s represent the portions of these quasi dominions already processed, to which the region function has assigned a priority with value strictly greater than p_s (see Item 1 above).

The initial state is composed of the priority function pr of the game and its maximal priority $\mathsf{pr}(\mathcal{D})$. Finally, we assume that a state s_1 is smaller than another state s_2 *w.r.t.* the partial order relation $\prec$, if the set of unprocessed positions in s_1 is a subset of those in s_2.

Definition 5 *(Search space).* A *search space* is a tuple $\mathcal{S} \triangleq \langle \mathrm{St}, s_I, \prec \rangle$, whose three components are defined as follows:

1. $\mathrm{St} \subseteq \mathrm{Rg} \times \mathrm{Pr}_\perp$ is the set *states* s, with $\mathsf{dom}(\mathsf{r}_s) = \mathrm{Ps}$;
2. $s_I \triangleq (\mathsf{pr}, \mathsf{pr}(\mathcal{D}))$ is the *initial state*;
3. $s_1 \prec s_2$ if $\mathrm{L}_{s_1} \subset \mathrm{L}_{s_2}$ or $\mathrm{L}_{s_1} = \mathrm{L}_{s_2} = \emptyset$ and $p_{s_1} = \perp < p_{s_2}$.

Note that the pseudo priority $\perp$ is used to indicate the situation where all positions have been processed, which corresponds to an empty local area.

Example 5. To exemplify the notion of state just defined, consider the game of the previous examples, which is reported again on the left-hand side of Fig. 2. Let us consider the state $s = (\mathsf{r}, 3)$, where $\mathsf{r} = \{\mathtt{a} \mapsto 0; \mathtt{c} \mapsto 1; \mathtt{e} \mapsto 3; \mathtt{d}, \mathtt{f}, \mathtt{h} \mapsto 6; \mathtt{b} \mapsto 7; \mathtt{g} \mapsto \top_0\}$ is the region function given in Example 4 and $\alpha_s = 1$. The local area L_s contains the positions a, c, and e. Among these positions, only e is part of the current region R_s. This position e, together with b, form the current quasi dominion Q_s. The quasi 0-dominion H^0_s contains positions d, f, g, and h, while the quasi 1-dominion H^1_s takes the only remaining position with priority greater than 3, namely b. Position g forms a 0-dominion on its own, represented in the picture by the solid blue line. Apart from this position, all the other ones are contained in open quasi dominions, indicated by dashed lines. For example, the set $\mathsf{r}^{-1}(6) = \{\mathtt{d}, \mathtt{f}, \mathtt{h}\}$ is a quasi 0-dominion. In fact, if Player 1 decides to remain inside, the adversary wins the play. However, Player 1 also has the choice to escape from position f by moving to e. A graphical representation of the state s is given on the right-hand side of Fig. 2, where blue-coloured (*resp.*, red-coloured) rectangles correspond to the slots of even (*resp.*, odd) priority in the region function r_s, dashed rectangles indicate that the corresponding set of positions is open and the grey-coloured area denotes the local area L_s.

During the exploration of the search space, a Priority Promotion algorithm typically traverses several types of states, some of which enjoy important properties that need to be explicitly identified, as they are exploited during the search for a dominion. In the following we introduce these types of states and explain what properties they hold. Identify the type of states it is crucial in order to prove the correctness of the approach.

Given a player $\alpha \in \mathbb{B}$, we say that a state $s \in \mathrm{St}$ is *α-maximal* if the quasi α-dominion H^α_s is α-maximal *w.r.t.* L_s. This means that the set of positions α-attracted by H^α_s from the local area L_s is empty, namely, $\mathrm{atr}^\alpha(\mathrm{H}^\alpha_s, \mathrm{L}_s) = \mathrm{H}^\alpha_s$. If s is α-maximal for both players $\alpha \in \mathbb{B}$, we simply say that it is *maximal*. It is useful to denote by $\mathrm{St}_{\mathrm{M}} \subseteq \mathrm{St}$ the corresponding

subset of states that are maximal and by $\supset_s \triangleq \supset \setminus (\mathrm{H}^0_s \cup \mathrm{H}^1_s) = \supset \setminus \{v \in \mathsf{dom}(\mathsf{r}_s) \mid \mathsf{r}_s(v) > p_s\}$ the induced subgame. A maximal state s is *strongly maximal* if the current region R_s is α_s-maximal *w.r.t.* L_s as well. We denote by St_{S}, which is a subset of St_{M}, the set of strongly maximal states.

Recall that the region R_s of a state s is contained in the current quasi dominion Q_s. We say that s is *open* if the player $\overline{\alpha}_s$ can escape from Q_s starting from R_s using a single move, *i.e.*, if $\mathrm{R}_s \cap \mathsf{esc}^{\overline{\alpha}_s}(\mathrm{Q}_s) \neq \emptyset$. In this case, the opponent may escape from R_s by either moving to the remaining portion of local area $\mathrm{L}_s \setminus \mathrm{R}_s$ or to the quasi $\overline{\alpha}_s$-dominion $\mathrm{H}^{\overline{\alpha}_s}_s$. The state is said to be *closed*, otherwise. For technical convenience, a state with an empty region is always considered open. Finally, a closed state s is *promotable*, if it is $\overline{\alpha}_s$-maximal and R_s is α_s-maximal *w.r.t.* L_s. By $\mathrm{St}_{\mathrm{P}} \subseteq \mathrm{St}$ we denote the set of promotable states. The crucial property of a promotable state s is the following: the only possible moves, if any, that player $\overline{\alpha}_s$ can use to escape from the α_s-region R_s necessarily lead to positions contained in the quasi α_s-dominions in $\mathrm{H}^{\alpha_s}_s$. Indeed, since s is $\overline{\alpha}_s$-maximal, and hence closed, no move from a $\overline{\alpha}_s$-position of $\mathrm{L}_s \supset \mathrm{R}_s$ can lead to $\mathrm{H}^{\overline{\alpha}_s}_s$, which means that all such moves remain inside the current quasi dominion Q_s. This is the property that is leveraged by the promotion operation at the basis of the approach.

Algorithm 1: RPP solver.

```
   function sol(s : St_M) : Rg
1  | if p_s ≠ ⊥ then
2  |   r_s ← r_s[atr^{α_s}_{⊃_s}(R_s) ↦ p_s]
3  |   if s is closed then
4  |     Promote(s)
   |   else
5  |     r_s ← sol(NextPr(s))
6  |     if s is closed then
7  |       Promote(s)
   |     else
8  |       Maximise(s)
9  |   r_s ← sol(s)
10 | return r_s
```

Auxiliary functions & procedures.

```
   function NextPr(s : St_S) : St_M
1  | q ← max(rng(r_s^{(<p_s)}))
2  | return (r_s, q)

   procedure Maximise(s : St)
1  | foreach α ∈ 𝔹 do
2  |   q ← min(rng(r_s ↾ H^α_s))
3  |   X ← atr^α(H^α_s, L_s)
4  |   r_s ← r_s[X ↦ q]
5  | r_s ← r_s[v ∈ L_s ↦ pr(v)]

   procedure Promote(s : St_P)
1  | q ← bep^{ᾱ_s}(R_s, r_s)
2  | r_s ← r_s[R_s ↦ q]
```

The main function **sol**, the auxiliary function NextPr (with no side-effects), and the two procedures Maximise and Promote (with side-effects on the state, *i.e.* the input state is modified in place) of the new Priority Promotion based approach, called *Recursive Priority Promotion* (RPP, for short), are provided in Algorithm 1.

The function **sol** assumes the input state s to be maximal, *i.e.*, $s \in \mathrm{St}_{\mathrm{M}}$. At Line 1 it checks whether there are still unprocessed positions in the game, that is, if the priority of the current state is different from the pseudo priority $\bot$. If this is the case, Line 2 maximises the region $\mathrm{R}_s = \mathsf{r}_s^{-1}(p_s)$ of the current state by computing its α_s-attractor $\mathsf{atr}^{\alpha_s}_{\supset_s}(\mathrm{R}_s)$, so that the resulting set is α_s-maximal. Then, the state s is made strongly maximal, by lifting those positions to priority p_s in the region function r_s, so that $s \in \mathrm{St}_{\mathrm{S}}$ at the end. If the resulting state s is closed (Line 3) it is also promotable, *i.e.*, $s \in \mathrm{St}_{\mathrm{P}}$, being maximal by hypothesis, and, therefore, a promotion is applied at Line 4 with a call to procedure Promote.

If, instead, s is open, which means that the opponent can escape from R_s by choosing a move exiting from the current quasi dominion Q_s, the algorithm proceeds to analyse the part of the game still unprocessed. To do this, the algorithm first computes the next state by means of NextPr(s), which simply identifies the next priority to consider, namely the maximum priority among the unprocessed positions. The resulting state is then passed to the recursive call at Line 5.

Once the recursive call completes, the state is updated with the new region function returned by the call. The new state s is such that the local area L_s coincides with R_s, since the recursive call terminates after all the previously unprocessed positions have been analysed. As a consequence, either the opponent cannot escape from R_s anymore or it can only move to its own quasi $\overline{\alpha}_s$-dominion $\mathrm{H}^{\overline{\alpha}_s}_s$.

Line 6 checks which one of the two possibilities occurs. In the first case, the new state s is closed, hence $\overline{\alpha}_s$-maximal. Moreover, since $\mathrm{L}_s = \mathrm{R}_s$, the region R_s cannot attract any other positions and is, therefore, α_s-maximal. This means that s is promotable, *i.e.*, $s \in \mathrm{St}_{\mathrm{P}}$, and Line 7 promotes the region to the quasi α_s-dominion $\mathrm{H}^{\alpha_s}_s$ inside the current quasi dominion Q_s. If, on the other hand, s is still open, then the opponent can escape to $\mathrm{H}^{\overline{\alpha}_s}_s$ from some positions in R_s. This means that the current state is not $\overline{\alpha}_s$-maximal and Line 8 fixes this by calling the procedure Maximise. The aim of this function is to reestablish maximality of the quasi dominions $\mathrm{H}^{\alpha_s}_s$ and $\mathrm{H}^{\overline{\alpha}_s}_s$ associated with the state s. This is done by allowing the processed quasi dominions recorded in the state, namely the ones with priority greater than p_s, to attract positions from the current region $\mathrm{R}_s = \mathrm{L}_s$. The surviving positions in R_s, if any, may not form a quasi α_s-dominion anymore and are reset to the original priorities recorded in the priority function pr of the game. In any case, when the computation reaches Line 9, whether coming from Line 4, Line 7, or Line 8, the state s is maximal, *i.e.*, $s \in \mathrm{St}_{\mathrm{M}}$, and a second, and final, recursive call is performed on s to process the remaining positions in L_s, if any (when coming from Line 7, L_s is necessarily empty, so the recursive call could be skipped).

The auxiliary function NextPr generates a new maximal state $\widehat{s} = \mathsf{NextPr}(s) \in \mathrm{St}_{\mathrm{M}}$, starting from a strongly-maximal one $s \in \mathrm{St}_{\mathrm{S}}$, that is lower in the partial order *w.r.t.* the one in input, *i.e.*, $\widehat{s} \prec s$. The state $\widehat{s}$ is obtained by changing the current

priority p_s to the highest priority q of the positions in $\mathsf{L}_s \setminus \mathsf{R}_s$. Observe that, when no such position exists, namely when $\mathsf{L}_s = \mathsf{R}_s$, the new priority coincides with the pseudo-priority $\bot$.

Maximise enforces the maximality property on the state s received as input, so that, in the resulting state obtained by modifying s in-place, no position of the local area L_s can be attracted by the quasi α-dominions H^{α}_s, with $\alpha \in \mathbb{B}$. To this end, the procedure computes (Line 2) the α-attractor $\mathsf{atr}^{\alpha}\big(\mathsf{H}^{\alpha}_s, \mathsf{L}_s\big)$, collecting all positions of L_s that player α can force to move into the attracting set H^{α}_s. These positions are then assigned (Line 4) the priority q corresponding to the minimum one associated with a position in the attracting set (Line 3). Since removing positions from the local area L_s may induce a violation of the two requirements of Definition 4, the positions that remain in L_s at the end of the for-each loop of Lines 1-4 need to be reset to their original priority recorded in pr, as prescribed at Line 5.

To conclude, the procedure Promote requires a promotable state $s \in \mathrm{St}_{\mathbb{P}}$ and applies a promotion operation to the region R_s, while preserving any maximality property already enjoyed by the input state. It first computes the opponent best-escape priority q for the set R_s *w.r.t.* r_s (Line 1). Intuitively, this is the smallest priority the opponent can reach with one move when escaping from the region R_s. Formally, it is defined as:

$$\mathsf{bep}^{\overline{\alpha}_s}(\mathsf{R}_s, \mathsf{r}_s) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r}_s \restriction \mathsf{rng}(I))),$$

where the binary relation $I \triangleq \mathsf{Mv}_{\mathcal{D}} \cap (\mathsf{esc}^{\overline{\alpha}_s}(\mathsf{R}_s) \times (\mathsf{dom}(\mathsf{r}_s) \setminus \mathsf{R}_s))$ contains all the moves leading outside R_s that the opponent can use to escape. The main property of promotable states observed above ensures that the best-escape priority q has always the same parity as α_s. The procedure, then, promotes R_s to q, by assigning (Line 2) the priority q to all the positions of R_s in the region function r_s. In particular, when the only possibility for player $\overline{\alpha}_s$ to escape from R_s is to reach $\mathsf{r}_s^{-1}(\top_\alpha)$, the value of q is $\top_\alpha$. In this case, we are promoting R_s from the status of quasi α_s-dominion to that of α_s-dominion. The correctness of this is ensured by Theorem 1.

At this point, by defining $\mathbf{sol}(\mathcal{D}) \triangleq (\mathsf{r}^{-1}(\top_0), \mathsf{r}^{-1}(\top_1))$, where $\mathsf{r} \triangleq \mathbf{sol}(s_I)$, we obtain a sound and complete solution algorithm for parity games. In particular, the soundness follows from the fact that RPP always traverses states having as invariant the property that $\mathsf{r}^{-1}(\top_0)$ and $\mathsf{r}^{-1}(\top_1)$ are dominions (see Item 2 of Definition 3). Completeness, instead, is due to the recursive nature of the algorithm, whose base case ensures that no position is left unprocessed at any given priority.

4. Parys' algorithm

Before obtaining a quasi-polynomial time algorithm based on the priority-promotion approach, we present here a reformulation of the idea of [56], where the recursion tree of the McNaughton algorithm is suitably truncated in order to avoid the exponential blowup.

Naturally, cutting some of the recursive calls may prevent us from deciding the winner for some of the positions with certainty. These intermediate results may thus contain positions with an *undetermined* status. The contribution of Parys' idea is the design of the truncation mechanism that offers bounded guarantees. Essentially, the undetermined sets contain all small dominions of one player and do not intersect with small dominions of the other. The *bounds* up to which these limited guarantees hold are shed quickly in the call tree: most of the calls are made with half precision, meaning that one of the bounds is halved, and only one is made with full precision, meaning that both bounds are kept. The result is a quasi-polynomial time solution that we will combine with the Priority Promotion approach in the next section, in order to preserve both the complexity of the first and the efficiency of the second.

A first step in the integration of Parys' approach with Priority Promotion is to formulate it in the same terms by introducing the required notation. The data structures are mostly inherited from the previous section. We need to extend the search space to represent also the precision bounds and the set of undetermined positions. While the two bounds can be simply encoded by natural numbers, the undetermined positions will be stored in a priority-lift function u that assigns to each priority $p > 0$ the set of undetermined positions returned by the recursive calls made at priority $p - 1$. A *Parys' state* s is a tuple $((\mathsf{r}, p), (\mathsf{u}, b_0, b_1))$, where the first pair comes from the search state define in Section 5 and the second tuple contains the function u, collecting the undetermined positions for each recursion level, and the two bounds b_0 and b_1. Recall that $\Delta^{\alpha}_{\mathsf{r}}$ (*resp.*, $\Delta^{\alpha}_{\mathsf{u}}$) collects the positions assigned to some α-priority by the function r (*resp.*, u), while $\Delta^{\alpha,p}_{\mathsf{r}} \subseteq \Delta^{\alpha}_{\mathsf{r}}$ (*resp.*, $\Delta^{\alpha,p}_{\mathsf{u}} \subseteq \Delta^{\alpha}_{\mathsf{u}}$) collects the positions assigned by r (*resp.*, u) to an α-priority of value at least p. For a state $s = ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1))$, we denote with $\mathsf{u}_s \triangleq \mathsf{u}$ the uncertain component, and with $b_{0s} \triangleq b_0$ and $b_{1s} \triangleq b_1$ the bounds of the state. Once again, denoted $\mathsf{r}_s \triangleq \mathsf{r}$ and $p_s \triangleq p$ the first two state components and by $\alpha_s \triangleq p_s \bmod 2$ the parity of the state, the local area $\mathsf{L}_s \triangleq \{v \in \mathsf{dom}(\mathsf{r}_s) \mid \mathsf{r}_s(v) \leq p_s\}$ of a state s contains the positions still unprocessed in that state, while $\mathsf{R}_s \triangleq \mathsf{r}^{-1}(p_s)$ denotes the region of s. Then, we set $\mathsf{Q}_s \triangleq \Delta^{\alpha_s,p_s}_{\mathsf{r}_s} \cup \Delta^{\overline{\alpha}_s,p_s}_{\mathsf{u}_s}$ and $\mathsf{H}^{\alpha}_s \triangleq (\Delta^{\alpha}_{\mathsf{r}_s} \cup \Delta^{\overline{\alpha}}_{\mathsf{u}_s}) \setminus \mathsf{L}_s$, for all $\alpha \in \mathbb{B}$. Essentially, H^0_s and H^1_s represent the portion of the game already processed to which the two functions r_s and u_s have assigned a priority with value strictly greater than p_s.

As opposed to the notions of RPP state, however, H^0_s and H^1_s are not necessarily quasi dominions, since they may include undetermined positions, namely those contained in $\Delta^{0,p_s}_{\mathsf{u}_s}$ and $\Delta^{1,p_s}_{\mathsf{u}_s}$. Only the subsets $\Delta^{0,q}_{\mathsf{r}_s}$ and $\Delta^{1,q}_{\mathsf{r}_s}$ are known to be quasi dominions, for all priorities q. The current subgame of a state s is, then, $\mathcal{D}_s \triangleq \mathcal{D} \setminus (\mathsf{H}^0_s \cup \mathsf{H}^1_s)$, containing all the unprocessed positions. In addition, $\mathsf{U}_s \triangleq \mathsf{u}_s^{-1}(p_s)$ collects the currently undetermined set of positions, which is required to satisfy the following property: it must contain all the $\overline{\alpha}_s$-dominions of size no greater than a given bound $b_{\overline{\alpha}_s}$ and it cannot intersect any α_s-dominion of size no greater than a second given bound b_{α_s}.

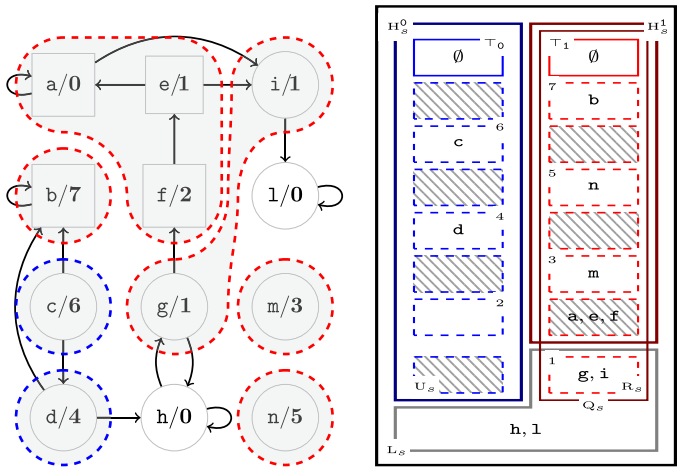

Fig. 3. A game and a corresponding state representation.

Definition 6 *(Parys' search space).* A *Parys' search space* is a tuple $\mathcal{S} \triangleq \langle \mathrm{St}, s_I, \prec \rangle$, whose three components are defined as follows:

1. $\mathrm{St} \subseteq (\mathrm{Rg} \times \mathrm{Pr}_\perp) \times (\mathrm{Pf} \times \mathbb{N} \times \mathbb{N})$ is the set of *Parys' states* s where:
   a) $\mathsf{dom}(\mathsf{r}_s) \cap \mathsf{dom}(\mathsf{u}_s) = \emptyset$, $\mathsf{dom}(\mathsf{r}_s) \cup \mathsf{dom}(\mathsf{u}_s) = \mathrm{Ps}$;
   b) $\{v \in \mathsf{dom}(\mathsf{u}_s) \mid \mathsf{u}_s(v) < p_s\} = \emptyset$ and $\mathsf{u}_s^{-1}(\top_\alpha) = \emptyset$, where $\alpha \triangleq \mathsf{pr}(\mathcal{D}) \bmod 2$;
   c) $\mathsf{dom}(\mathsf{u}_s) \cap \mathsf{esc}^{\overline{\alpha}}(\mathrm{H}_s^\alpha) = \emptyset$, for all $\alpha \in \mathbb{B}$;
2. $s_I \triangleq ((\mathsf{pr}, \mathsf{pr}(\mathcal{D})), (\varnothing, |\mathcal{D}|, |\mathcal{D}|))$ is the *initial state*;
3. $s_1 \prec s_2$ if $\mathrm{L}_{s_1} \subset \mathrm{L}_{s_2}$ or $\mathrm{L}_{s_1} = \mathrm{L}_{s_2} = \emptyset$ and $p_{s_1} = \perp < p_{s_2}$.

Item 1a ensures that the set of positions in the game is partitioned into two categories: those contained in the region function r_s, which are considered *determined*, in the sense that they belong to known quasi dominions; and those contained in the priority-lift function u_s, which are *undetermined*, since they form sets that only satisfy the bounded guarantees. Item 1b guarantees that there cannot be undermined positions at priorities lower than p_s and that the pseudo priority $\top_{\alpha_s}$ can never contain any undetermined position. These conditions capture the intuitions that all positions with priorities lower than the current one are still unprocessed, therefore they cannot be undetermined and that positions assigned to a top pseudo-priority must be determined as they are won by the corresponding player. Item 1c, instead, captures a more technical aspect of Parys' approach. Essentially, it requires that the player $\overline{\alpha}$ cannot escape to L_s starting from an undetermined position at any α-priority recursion level. Intuitively, the undetermined positions at any given level are formed by the subgame of recursive sub-calls with bound equal to one (depleted). Thus, they form closed sets, in the terminology of the previous section, from where the opponent cannot escape to lower priorities. For this reason, we shall refer to this property also as *closed under attraction*. The initial state is composed of the pair of the priority function pr and the maximal priority $\mathsf{pr}(\mathcal{D})$ of the game, and the tuple with the empty priority-lift function $\varnothing$, and the two (full) bounds that corresponding to number of positions in the game. Finally, the order relation $\prec$, is defined the same way as done for the state space for Priority Promotion.

Example 6. Fig. 3 depicts an example game on the left-side and, on the right-side, the corresponding graphical representation of one of the states computed during the execution of Parys' algorithm, whose interpretation is similar to that of Fig. 2 and where, in addition, we indicate with pattern-filled rectangles the slots of the undetermined function u_s and the undetermined region U_s. Let us consider the state $s = ((\mathsf{r}, 1), (\mathsf{u}, 2, 2))$, where $\mathsf{r} = \{\mathtt{b} \mapsto 7; \mathtt{c} \mapsto 6; \mathtt{n} \mapsto 5; \mathtt{d} \mapsto 4; \mathtt{m} \mapsto 3; \mathtt{g}, \mathtt{i} \mapsto 1\}$ is the region function, $\mathsf{u} = \{\mathtt{a}, \mathtt{e}, \mathtt{f} \mapsto 2\}$ is the undetermined function, and $\alpha_s = 1$. The current priority p_s is 1 and the two bounds are $b_{0s} = 2$ and $b_{1s} = 2$. The local area L_s contains the positions $\mathtt{g}, \mathtt{h}, \mathtt{i}$, and $\mathtt{l}$. Among these, $\mathtt{g}$ and $\mathtt{i}$ are part of the current region R_s. These positions, together with $\Delta_{\mathsf{u}}^1 = \{\mathtt{a}, \mathtt{e}, \mathtt{f}\}$, and $\mathtt{b}, \mathtt{n}, \mathtt{m}$, form the current set Q_s. The quasi $\mathtt{0}$-dominion H_s^0 contains positions $\mathtt{d}$, and $\mathtt{c}$, which are both part of the region function r. The quasi $\mathtt{1}$-dominion H_s^1, on the other hand, takes the remaining positions with priority greater than 1, namely positions $\mathtt{b}, \mathtt{n}$ and $\mathtt{m}$, stored in the region function r, and positions $\mathtt{a}, \mathtt{e}$ and $\mathtt{f}$, which are stored in the uncertain function u.

Just like RPP does not use undetermined positions, Parys' approach does not use promotions, even if it handles regions (by means of the attractor of the positions with highest priority). Consequently, little happens to the region function in our

representation of Parys' algorithm: positions that are added to U_s are removed from the region function, and when U_s is destroyed, they are added (with their native priorities) back to r_s.

Since the purpose of this section is to revisit Parys' algorithm, here we only outline the principles, by reporting some of the standard lemmas from [59] that will be used later on also by the hybrid approach.

The first lemma simply states that dominions are closed under opponent attraction from the complement set of its positions.

Lemma 1. *Let* D *be a dominion for player* α *and* $\mathrm{S} \subseteq \mathrm{Ps}$ *a set of positions. If* D *does not intersect with* S *then it does not intersect with* $\mathsf{atr}^{\overline{\alpha}}(\mathrm{S})$ *either.*

The second specifies that in case a dominion includes positions with a higher opponent priority, then it also has to include a smaller dominion closed under opponent attraction from the set of positions with the higher opponent priority.

Lemma 2. *Let* D *be a dominion for player* α *with highest priority* p *and* R *a* $\overline{\alpha}$*-region of priority* q *with* $q > p$*. If* D *intersects with* R *then it contains a non-empty sub-dominion that does not intersect with* $\mathsf{atr}^{\overline{\alpha}}(\mathrm{R})$.

Algorithm 2: Parys solver.

```
   function sol(s : St) : 2^Ps
1  if p_s = ⊥ ∨ b_{α_s} = 1 then
2  |  return (r_s, u_s)
   else
3  |  hsol(s)
4  |  ŝ ← s
5  |  (r_s, u_s) ← sol(NextPr(s))
6  |  Maximise(s)
7  |  if s ≺ ŝ then hsol(s)
8  |  return Und(s)
```

Auxiliary functions & procedures.

```
   function NextPr(s : St) : St
1  |  return ((r_s, p_s − 1), (u_s, p_s, b_{0s}, b_{1s}))

   function Half(s : St) : St
1  |  (b_{0s}, b_{1s}) ← (⌊b_{0s}/(1+α_s)⌋, ⌊b_{1s}/(2−α_s)⌋)
2  |  return NextPr(s)
```

Half-solver.

```
   procedure hsol(s : St)
1  |  repeat
2  |  |  r_s ← r_s[atr^{α_s}_{⊃_s}(R_s) ↦ p_s]
3  |  |  ŝ ← s
4  |  |  (r_s, u_s) ← sol(Half(s))
5  |  |  Maximise(s)
   |  until s ⊀ ŝ
```

Auxiliary functions & procedures.

```
   procedure Maximise(s : St)
1  |  X ← atr^{ᾱ_s}(H^{ᾱ_s}_s, L_s)
2  |  r_s ← r_s \ X
3  |  u_s ← u_s[X ↦ p_s]
4  |  if X ≠ ∅ then
5  |  |  r_s ← r_s[v ∈ R_s ↦ pr(v)]

   function Und(s : St) : Rg × Pf
1  |  u_s ← u_s[L_s ↦ p_s + 1]
2  |  r_s ← r_s^{(> p_s)}[v ∈ U_s ↦ pr(v)]
3  |  return (r_s, u_s)
```

The main algorithm is composed of a full bound precision function **sol** and an halved precision procedure **hsol** that are reported in the upper part of Algorithm 2. The auxiliary procedure Maximise and the auxiliary functions NextPr, Half, and Und are reported in the lower part.

In order to outline the structure of function **sol**, it is helpful to first describe the halved precision procedure **hsol**. The half-solver **hsol** maximises, at Line 2, the region $\mathrm{R}_s = \mathrm{r}_s^{-1}(p_s)$ of the current state by computing its α_s-attractor $\mathsf{atr}^{\alpha_s}_{\supset_s}(\mathrm{R}_s)$. The remaining subgame from which R_s has been removed, is recursively solved by a call of **sol** with half precision by means of the function Half that halves the bound b_{α_s} for the current player α_s. As long as the maximisation of the state returned by the call to **sol** at Line 4 differs from the copy of the local state saved at Line 3, the procedure will repeat the loop of Lines 2-5. Note that in all these iterations the priority p_s remains the same; therefore, the positions in current region R_s have to change at every iteration (and they can only do that by shrinking), otherwise the repeat-until loop will stop. Then this loop ensures that every small $\overline{\alpha}_s$-dominion, those of size at most $\left\lfloor \frac{\mathtt{b}_{\overline{\alpha}_s}}{2} \right\rfloor$ in the subgame that are not included in small α_s-regions of size at most $\mathtt{b}_{\alpha_s}$, are found and returned in the uncertain function u_s. The observation above is a consequence of Lemmas 1 and 2. If such a small $\overline{\alpha}_s$-dominion does not intersect with the current α_s-region R_s, then by Lemma 1, it is included in the remaining subgame, and identified by the call of **sol**. If, instead, it intersects with R_s, then, by Lemma 2, there must be a sub-dominion that cannot be attracted by R_s and, therefore, will be identified by the next call to **sol**.

The function **sol** has a base case at Line 1, where it checks if there are unprocessed positions in the current state (which implies that $p_s > \bot$) or the bound for player α_s is not depleted. The function can perform three recursive calls, two of the half-solver, and one of **sol** itself. The first call to **hsol**, at Line 3, serves to find small dominions in the subgame, while the call to **sol**, at Line 5, finds big dominions that the previous call might have been unable to find due to the bound constraint. If the full precision call has not found a $\overline{\alpha}_s$-dominion, then the subgame does not contain any dominion for the bound available. In this case a second call to **hsol** would be pointless as **hsol** has a smaller bound than **sol**. Therefore, the function ends by calling Und, which records the entire current subgame L_s as the undetermined set for the caller at level $p_s + 1$.

Otherwise, when the full precision call returns a new $\overline{\alpha}_s$-dominion D, there might be other small $\overline{\alpha}_s$-dominions in the subgame to find. This follows for the fact that D is bigger than $\left\lfloor \frac{\flat_{\overline{\alpha}_s}}{2} \right\rfloor$ and it has been closed under $\overline{\alpha}_s$-attractor by the call to Maximise at Line 6. This operation can reset R_s that has been identified by **hsol** and possibly release a small $\overline{\alpha}_s$-dominion previously included in R_s. For this reason a second and final call to **hsol** is performed to identify such small $\overline{\alpha}_s$-dominions. This case occurs only when the updated state returned at Line 5 differs from the copy of the local state saved at Line 4. At the end of the function, when Und is called at Line 8, all $\overline{\alpha}_s$-dominions of size at most $\flat_{\overline{\alpha}_s}$ have been identified and processed in U_s. Therefore, none of them intersects with L_s that instead contains the bounded winning region of player α_s. At this point, since the guarantees for the winning α_s-region are bounded, L_s is moved to the uncertain set U of the caller priority, that is $p_s + 1$.

The auxiliary function NextPr generates a new maximal state by decreasing the current priority by 1. Also Half generates a new maximal state, by first halving the precision bound for player α_s and then calling NextPr.

As in the case of the RPP solver, Maximise enforces the maximality property on the current state s. As a result, in the modifies state s, no positions of the local area L_s can be attracted by the quasi $\overline{\alpha}_s$-dominion $\mathsf{H}_s^{\overline{\alpha}_s}$. To obtain maximality, the procedure computes at Line 1 the $\overline{\alpha}_s$-attractor $\mathsf{atr}^{\overline{\alpha}_s}\left(\mathsf{H}_s^{\overline{\alpha}_s}, \mathsf{L}_s\right)$, which collects all the position in L_s that player $\overline{\alpha}_s$ can force to move into $\mathsf{H}_s^{\overline{\alpha}_s}$. The attracted set is then removed from r_s at Line 2 and assigned at Line 3 to u_s at the current priority p_s. Attracting positions from L_s may result in the violation of the requirements of Definition 4 for the positions that remain in R_s. For this reason, if $\mathsf{H}_s^{\overline{\alpha}_s}$ does attract some positions from L_s, the rest of R_s is reset, *i.e.* the priority of its positions is set to their original priority in pr, as prescribed by Line 5. Note that the remaining positions of L_s need not be reset, since their priorities are already those recorded in pr.

Finally, the function Und takes care of handling the undetermined positions computed at the end of a call to the solver. It receives in input a state where all $\overline{\alpha}_s$-dominions of size at most $\flat_{\overline{\alpha}_s}$ (that are not included in α_s-regions of size at most $\flat_{\alpha_s}$) have been moved into U_s. It follows that L_s is a bounded winning region for player α_s.

To explain how Und works, let s be the current state and s' the one of the caller priority level $p_s + 1$, where, in case $p_s = \mathsf{pr}(\mathcal{D})$, then $p_s + 1$ denotes the top pseudo-priority $\top_{\overline{\alpha}}$, with $\mathsf{pr}(\mathcal{D}) \equiv_2 \alpha$. Since Und is the last operation executed before the return to the call corresponding to state s', the bounded winning α_s-region L_s at the current priority level p_s is added to the uncertain winning set of positions for player $\overline{\alpha}_s$ in s', that is $\mathsf{U}_{s'}$ (Line 1). The positions of U_s are instead reset at Line 2 and moved into r_s at their original priority. Therefore, these positions will be part of the unprocessed partition of the game of s', that is $\mathsf{L}_{s'}$. Clearly, no new position added to the undetermined function can be forced to move to $\mathsf{L}_{s'}$ by means of an opponent attractor operation in accordance with Item 1c of Definition 6 applied to the new state s'.

When $\mathbf{sol}(s_I)$ terminates by calling Und, it returns the pair composed of r_{s_I} and u_{s_I}, and the two winning regions correspond to the sets $\mathsf{Wn}_{\mathcal{D}}^{\alpha} = \mathsf{u}_{s_I}^{-1}(\top_{\overline{\alpha}})$ and $\mathsf{Wn}_{\mathcal{D}}^{\overline{\alpha}} = \mathcal{D} \setminus \mathsf{Wn}_{\mathcal{D}}^{\alpha}$.

Corollary 1. *Let* (r, u) *be the pair of region and promotion functions computed by Algorithm 2 on a game* $\mathcal{D}$*. The winning region for player* α *corresponds to* $\mathsf{u}^{-1}(\top_{\overline{\alpha}})$. □

We provided a different formalisation of Parys' algorithm by means of the pair of region and promotion functions. The correctness corollary can be found, in a different formulation, in the original paper [56].

5. A hybrid algorithm

In our hybrid approach, we have to synthesise the use of regions employed in the RPP solver and the use of undetermined sets introduced to implement Parys' approach. To formalise this intuition into a hybrid state, we need to further modify the simple state of RPP *w.r.t.* the changes already introduced in the Parys' search space of Definition 6. The good news is that the data structure of a Parys' state stores almost all the information required by the hybrid algorithm. It follows that, syntactically, the hybrid state corresponds to the Parys' one with an additional priority and, semantically, it satisfies an additional property.

The most relevant difference between the two approaches is that, unlike Parys' algorithm, the hybrid algorithm does not explore sequences of states with consecutive priorities. Instead, like the RPP approach, it can skip priorities that do not occur in the current subgame. To account for the gap between the priority of the parent recursive call and the current one associated with the state, we introduce in the state an additional component c that records the caller priority. Clearly, since no recursion levels are present between the caller and the current priorities, no position can be recorded between those two levels in either of the two functions r and u of that state. This follows from the fact that the priority selected by the caller for the next sub-call is the maximum one existing in the subgame. Therefore, both r and u will not contain positions with priority lower that the caller one and higher than the sub-caller one.

A *hybrid state* s is a tuple $((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$, which contains all the components of the Parys' state, namely, the region function r, the current priority p, the undetermined function u, and the two bounds b_0 and b_1, one for each player, as well as the additional priority c of the caller. We set $c_s \triangleq c$, while the elements α_s, p_s, $\flat_{\alpha_s}$, r_s, u_s, Q_s, H_s^{α}, U_s, R_s, and L_s are defined as in the previous section. The caller priority of the initial state is set to the pseudo priority $\top_{\overline{\alpha_s}}$.

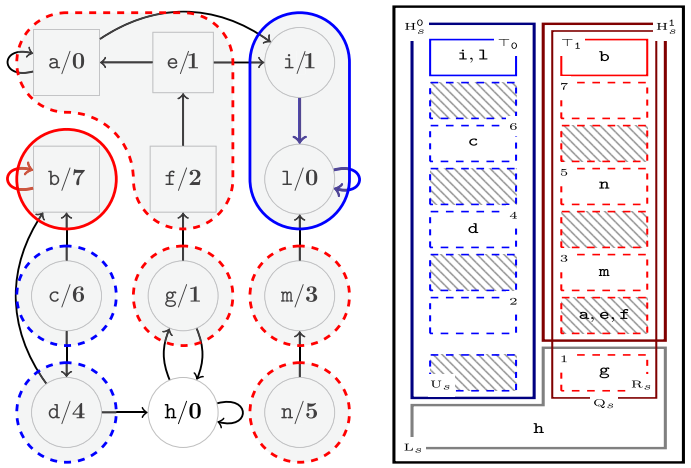

Fig. 4. A game and a corresponding state representation.

Definition 7 *(Hybrid search space).* A *hybrid search space* is a tuple $\mathcal{S} \triangleq \langle \mathrm{St}, s_I, \prec \rangle$, whose three components are defined as follows:

1. $\mathrm{St} \subseteq (\mathrm{Rg} \times \mathrm{Pr}_\bot) \times (\mathrm{Pf} \times \mathbb{N} \times \mathbb{N}) \times \mathrm{Pr}_\top$ is the set of *hybrid states* s where:
   a) $\mathsf{dom}(\mathsf{r}_s) \cap \mathsf{dom}(\mathsf{u}_s) = \emptyset$, $\mathsf{dom}(\mathsf{r}_s) \cup \mathsf{dom}(\mathsf{u}_s) = \mathrm{Ps}$;
   b) $\{v \in \mathsf{dom}(\mathsf{u}_s) \mid \mathsf{u}_s(v) < p_s\} = \emptyset$ and $\mathsf{u}_s^{-1}(\top_\alpha) = \emptyset$, where $\alpha \triangleq \mathsf{pr}(\eth) \bmod 2$;
   c) $\mathsf{dom}(\mathsf{u}_s) \cap \mathsf{esc}^{\overline{\alpha}}(\mathrm{H}_s^\alpha) = \emptyset$, for all $\alpha \in \mathbb{B}$;
   d) $\{v \in \mathsf{dom}(\mathsf{f}) \mid p_s < \mathsf{f}(v) < c_s\} = \emptyset$, for $\mathsf{f} \in \{\mathsf{r}_s, \mathsf{u}_s\}$, and $p_s < c_s$;
2. $s_I \triangleq ((\mathsf{pr}, \mathsf{pr}(\eth)), (\varnothing, |\eth|, |\eth|), \top_{\overline{\alpha}})$ is the *initial state*, where $\alpha \triangleq \mathsf{pr}(\eth) \bmod 2$;
3. $s_1 \prec s_2$ if $\mathrm{L}_{s_1} \subset \mathrm{L}_{s_2}$ or $\mathrm{L}_{s_1} = \mathrm{L}_{s_2} = \emptyset$ and $p_{s_1} = \bot < p_{s_2}$.

The only new condition *w.r.t.* Definition 6 is Item 1d, which requires that no position be contained in either r_s or u_s at priorities between p_s, the current priority, and c_s, the caller one.

Most of the concepts and notation introduced in a Parys' state have a similar meaning and play a similar role in a hybrid state. In particular, given a hybrid state $s \in \mathrm{St}$, the set L_s identifies the *local area*, *i.e.*, the set of positions yet to analyse, while R_s is the quasi α_s-dominion, called *region*, included in L_s, which the algorithm is currently focusing on. Moreover, the two sets H_s^0 and H_s^1 partition the portion of game already processed.

Example 7. Fig. 4 depicts a snapshot of the execution of the hybrid algorithm on the example game described in Example 6. Again, the left-side depicts the game, while the right-side provides a graphical representation of the hybrid state in the considered snapshot with the same graphical conventions adopted in Fig. 3. The hybrid state s corresponds to the tuple $((\mathsf{r}, 1), (\mathsf{u}, 2, 1), 1)$, where the first two components are region function $\mathsf{r} = \{\mathtt{i}, \mathtt{l} \mapsto \top_0; \mathtt{b} \mapsto \top_1; \mathtt{c} \mapsto 6; \mathtt{n} \mapsto 5; \mathtt{d} \mapsto 4; \mathtt{n} \mapsto 3; \mathtt{g} \mapsto 1\}$ and the current priority $p = 1$. It follows that the parity of the state is $\alpha_s = 1$. In the second part of the state we have $\mathsf{u} = \{\mathtt{a}, \mathtt{e}, \mathtt{f} \mapsto 2\}$ that is the undetermined function, the two bounds b_0 and b_1, that are both 1, and the caller priority c set to 2. The local area L_s, represented in grey at the bottom of the picture, contains the positions `g`, and `h`. Among these, only `g` is part of the current region R_s. This position, together with $\Delta_{\mathsf{u}}^1 = \{\mathtt{a}, \mathtt{e}, \mathtt{f}\}$ and $\Delta_{\mathsf{r}}^1 = \{\mathtt{b}, \mathtt{n}, \mathtt{m}\}$ forms the current quasi dominion Q_s represented by the right column in red of the picture. The quasi `0`-dominion H_s^0, represented in blue, contains positions `i`, `l`, `c`, and `d`, that are all stored in the region function r, while the quasi `1`-dominion H_s^1 takes the remaining positions with priority greater than 1. The positions stored in r with value $\top_0$ or $\top_1$ are dominion already identified, and therefore solved by the algorithm. For this reason, in the picture, these positions are represented with non-dashed lines and their winning strategy is highlighted with coloured edges.

It is worth noticing that Parys' algorithm reaches the state depicted in Example 6 regardless the two positions `m`, and `n`. On the contrary, for the hybrid algorithm, these two positions are required to deplete the precision bound. Without `m` and `n`, the hybrid algorithm can easily solve the game thanks to the skip of the priorities that do not occur in the subgame. It follows that whenever a game has priorities that are not consecutive, the hybrid algorithm solves it easier than Parys' approach.

Given a player $\alpha \in \mathbb{B}$, we say that a hybrid state $s \in \mathrm{St}$ is α*-maximal*, if the quasi α-dominion H_s^α is α-maximal *w.r.t.* L_s. If s is α-maximal *w.r.t.* both players $\alpha \in \mathbb{B}$, we say that it is *maximal*. We denote with $\mathrm{St}_{\mathrm{M}} \subseteq \mathrm{St}$ the set of maximal hybrid

states and with $\supset_s \triangleq \supset \setminus (\mathrm{H}^0_s \cup \mathrm{H}^1_s)$ the induced subgame of s. A maximal hybrid state s is *strongly maximal*, if the current region R_s is α_s-maximal *w.r.t.* L_s. With $\mathrm{St}_{\mathrm{S}} \subseteq \mathrm{St}_{\mathrm{M}}$ we denote the set of strongly maximal hybrid states. Again, we say that s is *open* if $\mathrm{R}_s \cap \mathsf{esc}^{\overline{\alpha}_s}(\mathrm{Q}_s) \neq \emptyset$, and that it is *closed*, otherwise. For technical convenience, we consider open a hybrid state whose region R_s is empty. Finally, a closed hybrid state s is *promotable*, if it is $\overline{\alpha}_s$-maximal and R_s is α_s-maximal *w.r.t.* L_s. With $\mathrm{St}_{\mathrm{P}} \subseteq \mathrm{St}$ we denote the set of promotable hybrid states.

The main functions and the auxiliaries procedures of the *Hybrid Priority Promotion algorithm* (HPP, for short), which are provided in Algorithm 5, synthesise the recursive Priority Promotion technique of Algorithm 1 and the recursion-tree truncation idea of Algorithm 2 into a single approach.

Algorithm 5: HPP solver.

```
   function sol(s : St_M) : Rg × Pf
1  | if p_s = ⊥ ∨ b_α = 0 then
2  | | return (r_s, u_s)
   | else
3  | | hsol(s)
4  | | ŝ ← s
5  | | (r_s, u_s) ← sol(NextPr(s))
6  | | if s is open then
7  | | | Maximise(s)
   | | else
8  | | | Promote(s)
9  | | if s ≺ ŝ then hsol(s)
10 | | return Und(s)
```

Algorithm 6: Half-solver.

```
   procedure hsol(s : St_M)
1  | repeat
2  | | r_s ← r_s[atr^{α_s}_{⊃_s}(R_s) ↦ p_s]
3  | | ŝ ← s
4  | | if s is open then
5  | | | (r_s, u_s) ← sol(Half(s))
6  | | | if s is open then
7  | | | | Maximise(s)
   | | | else
8  | | | | Promote(s)
   | | else
9  | | | Promote(s)
   | until s ⊀ ŝ
```

Auxiliary functions.

```
   function NextPr(s : St_S) : St_M
1  | q ← max(rng(r_s^{(<p_s)}))
2  | return ((r_s, q), (u_s, p_s, b_{0s}, b_{1s}))

   function Half(s : St_S) : St_M
1  | (b_{0s}, b_{1s}) ← (⌊b_{0s}/(1+α_s)⌋, ⌊b_{1s}/(2−α_s)⌋)
2  | return NextPr(s)

   function Und(s : St_S) : Rg × Pf
1  | if c_s ≡_2 α_s then
2  | | u_s ← u_s[U_s ↦ c_s]
   | else
3  | | u_s ← u_s^{(≥c_s)}[L_s ↦ c_s]
4  | | r_s ← r_s^{(≥c_s)}[v∈U_s ↦ pr(v)]
5  | return (r_s, u_s)
```

Auxiliary procedures.

```
   procedure Promote(s : St_P)
1  | (p_r, p_u) ← (bep^{ᾱ_s}(R_s, r_s), bep^{ᾱ_s}(R_s, u_s))
2  | if p_r ≤ p_u then
3  | | r_s ← r_s[R_s ↦ p_r]
   | else
4  | | (r_s, u_s) ← (r_s \ R_s, u_s[R_s ↦ p_u])

   procedure Maximise(s : St)
1  | Z ← R_s
2  | foreach α ∈ 𝔹 do
3  | | X ← atr^α(H^α_s, L_s)
4  | | q ← min(rng((r_s ∪ u_s) ↾ H^α_s))
5  | | if q ≡_2 α then
6  | | | r_s ← r_s[X ↦ q]
7  | | | u_s ← u_s \ X
   | | else
8  | | | r_s ← r_s \ X
9  | | | u_s ← u_s[X ↦ q]
10 | if Z ≠ R_s then
11 | | r_s ← r_s[v∈R_s ↦ pr(v)]
```

Like RPP, the main function **sol** assumes the input state s to be maximal, *i.e.*, $s \in \mathrm{St}_{\mathrm{M}}$. Line 1 checks whether (i) there are no unprocessed positions in the game or (ii) the bound for player $\alpha[s]$ for the guarantees on the undetermined positions has reached the threshold zero. If one of these conditions is satisfied, the current region function r_s and the undetermined function u_s are returned unmodified at Line 2, as no further progress can be obtained in the current recursive call. Otherwise, similar to Parys' approach, the search for a dominion is split into three phases: (i) a first search with halved precision made by calling the auxiliary mutually-recursive procedure **hsol** (Line 3); (ii) a second search with full precision with a recursive call to **sol** itself (Lines 4 to 8); (iii) a final search by means of **hsol**, again with halved precision, conditioned to the actual progress obtained during the previous phase (Line 9). On termination of these three phases, the information about the undetermined positions contained in the local area L_s or in the undetermined set U_s are handled by the function Und at Line 10.

To discuss the guarantees and their effects in more detail, let us fix a small $\overline{\alpha}$-dominion D, with $|\mathrm{D}| \leq b_{\overline{\alpha}_s}$. Then, the call to **hsol** at Line 3 modifies the maximal state s given as input in-place, turning it into a strongly-maximal one so that L_s no longer contains any tiny dominions of player $\overline{\alpha}_s$ of size less than or equal to $\lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$. It follows that **hsol** might have not enough precision bound to guarantee that D is found. Therefore, what eventually remains is a set $\mathrm{D}' = \mathrm{D} \cap \mathrm{L}_s$ that is a dominion in L_s by Lemma 1. The set $\mathrm{D} \setminus \mathrm{D}'$, has been instead processed and added either to u_s at some priority $q \geq p$ or to r_s at some priority $q > p$. Moreover, by Lemma 2, D' has a non-empty sub-dominion D'' that does not intersect with the current region R_s. Since L_s contains no tiny $\overline{\alpha}_s$-dominions, it holds that $|\mathrm{D}''| > \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$, and $\mathrm{D}' \setminus \mathrm{D}'' < \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$. After this first call

to **hsol**, the resulting state is locally stored, at Line 4, in order to determine, later on, whether the second phase achieves any progress.

The algorithm then proceeds to analyse the remaining part of the game still unprocessed. To do so, the next state computed by NextPr(s) is given as input to the recursive call **sol** at Line 5. Once the call completes, the state is updated with the two new functions returned by the call. At this point, also D'' has been processed, since **sol** has enough precision bound to find D''. What eventually remains unsolved is the set $D' \setminus D''$. This set, however can only contain tiny $\overline{\alpha}_s$-dominions that can also be found by **hsol**.

At this point, depending on whether or not the state is closed, either a promotion or a maximisation operation is performed (Lines 6 to 8) to ensure that the new state is maximal. If the middle phase with full precision made some progress in the search, a last call to **hsol** at Line 9 is performed, which again modifies the current state in-place turning it into a strongly-maximal one. Due to the small size of the residual unsolved dominions, this call suffices to solve the corresponding subgame ∂_s. If no progress occurred, instead, the current state s is equal to the one stored in and previously returned by the first call to **hsol**. Thus, s is strongly-maximal, D' was empty in the first place, and D has been completely processed in that first call to **hsol**. In either case, the state is fed to the function Und to handle possible undetermined positions, after which the current call terminates.

The procedure **hsol** simply executes the main body of the RPP algorithm by making mutually-recursive calls to the function **sol** (Line 5) with halved precision until no progress on the search for a dominion can be made. Similarly to RPP, the auxiliary function NextPr identifies the next priority to consider, and the promotion and maximisation procedures, Promote and Maximise, generalise the corresponding procedures defined for RPP. Specifically, Promote applies a promotion, but this time the best escape priority can belong either to r or to u. Therefore, in base of the escaping priority, the current region R can be promoted to r, as for the classic Priority Promotion approach, or removed for it and promoted to u. Finally, Maximise turns a state s into a $\overline{\alpha_s}$-maximal one, by maximising both r and u. Similar to Parys' algorithm, the Half function halves the bound of the player $\overline{\alpha}_s$, leaving the bound of player α_s unchanged. Finally, the Und function starts from a strongly maximal state $s \in \mathrm{St}_\boxtimes$ where all small dominions of player $\overline{\alpha}_s$ of size $\leq b_{\overline{\alpha}_s}$ *at the time of the current call* are processed. Thus, neither do any of the small dominions ($\leq b_{\overline{\alpha}_s}$) of player $\overline{\alpha}_s$ intersect with U_s, nor do any of the small dominions ($\leq b_{\overline{\alpha}_s}$) of player $\overline{\alpha}_s$ intersect with L_s. Depending on the parity of the calling priority c_s, we can return the respective set (U_s or L_s) and, where the parity is different, reset the positions of U_s in r to their original priority.

At this point, by defining the winning regions of the players as $\mathrm{Wn}^{\alpha}_{\partial} = \mathsf{r}^{-1}(\top_\alpha) \cup \mathsf{u}^{-1}(\top_{\overline{\alpha}})$ and $\mathrm{Wn}^{\overline{\alpha}}_{\partial} = \partial \setminus \mathrm{Wn}^{\alpha}_{\partial}$, where $(\mathsf{r}, \mathsf{u}) \triangleq \mathbf{sol}(s_I)$, we obtain a sound and complete solution algorithm, whose time-complexity is quasi-polynomial, as we shall show in the next section.

6. Correctness and complexity

We now discuss how we can entangle the concepts of Priority Promotion, *i.e.* the transfer of information across the call structure that makes it more efficient in practice, with the concept of relative guarantees that provides favourable complexity bounds to Parys' algorithm. Before turning to the principle guarantees provided by the algorithm, we note that the two algorithms from the previous sections (Parys' and the selected variation of Priority Promotion) can be viewed as variations of our hybrid algorithm. This is particularly easy to see for the RPP algorithm from Section 3: when we set the bounds to infinity (or to 2^c, where c is the number of different priorities of the game) then the algorithm never runs out of bounds. In this case, the function u is never used, and the algorithm behaves exactly as Algorithm 1. In more detail, the whole game is recursively solved by the first call to **hsol** at Line 3 of Algorithm 5 that act like **sol** of Algorithm 1. Half has no effect, because the bounds always remain greater than 0, and Und does nothing since U_s is constantly empty and, at the end of **sol**, L_s is empty as well. Finally, Promote and Maximise only use the function r as a consequence of the fact that u is never filled by Und.

The connection to Parys' algorithm is slightly looser, but essentially it works one priority at a time solving the game purely with attractors. Hence it gives up promoting positions to higher priorities. This change does not impact on the partial correctness argument as it guarantees progress, but significantly slow down the solution procedure. In more detail, neither **sol** nor **hsol** check for open/closed regions as all regions are always treated as open. Half is the same, NextPr does not skip any priority and, as a consequence, u always applies the case in which the parity of the current priority is different from the parity of the caller priority, assigning the uncertain positions to the previous (caller) priority. Finally, Maximise closes only u under attractor.

As the algorithm is a hybrid one, its correctness proof has both local and global aspects. The global guarantees are that the regions stored in $\mathsf{r}^{(>p_s)}$ and the bounded dominions stored in $\mathsf{u}^{(\geq p_s)}$ retain their properties (that are described in the last lemma) in all function calls. The local guarantees refer instead to the local area L_s of the current state s introduced in Definition 7. To conveniently reason about the guarantees we define the set P_s, which stores the local area L_s at the beginning of the call of **sol**; P_s is then available also in the **hsol** at the level where they are called.

This additional set P_s of initial positions is relevant as the guarantees of finding small dominions is formulated relative to it and not relative to the local area L_s at the end of **sol**. The correctness proof falls into lemmas that refer to the guarantees maintained by the auxiliary functions, and an inductive proof of the main lemma. We introduce in the following the lemmas for the auxiliary functions, while their proofs are reported in Appendix A. A final one, instead, outline the inductive proof

for the main functions making use of the previous lemmas. After them, we report a theorem for the correctness of the algorithm that immediately follows from the invariants that have been proved before.

Lemma 3. *Given a strongly-maximal hybrid state $s \in \mathrm{St}_{\mathrm{S}}$, the functions* $\mathsf{NextPr}(s)$ *and* $\mathsf{Half}(s)$ *return a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \prec s$.*

This lemma states that both NextPr and Half take as input a strongly-maximal hybrid state and return a new maximal hybrid state which is smaller *w.r.t.* $\prec$ (the ordering among states from Definition 7).

Lemma 4. *Given a hybrid state $s \in \mathrm{St}$, the procedure* $\mathsf{Maximise}(s)$ *modifies in-place s into a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \preccurlyeq s$.*

The procedure Maximise takes as input a hybrid state that modifies in-place into a maximal hybrid state which is no greater than the original one *w.r.t.* $\prec$.

Lemma 5. *Given a promotable hybrid state $s \in \mathrm{St}_{\mathrm{P}}$, the procedure* $\mathsf{Promote}(s)$ *modifies in-place s into a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \prec s$.*

The procedure Promote takes as input a promotable hybrid state that modifies in-place into a maximal hybrid state which is smaller than the original one *w.r.t.* $\prec$.

Lemma 6. *Given a strongly-maximal hybrid state $s \in \mathrm{St}_{\mathrm{S}}$, the function* $\mathsf{Und}(s)$ *returns a pair $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \in \mathrm{Rg} \times \mathrm{Pf}$ of a region function $\widehat{\mathsf{r}}$ and a promotion function $\widehat{\mathsf{u}}$ such that: 1) $\mathsf{r}_s^{(>p_s)} \subseteq \widehat{\mathsf{r}}^{(>p_s)}$; 2) $\mathsf{u}_s^{(>p_s)} \subseteq \widehat{\mathsf{u}}^{(>p_s)}$; 3) $\widehat{s} = ((\widehat{\mathsf{r}}, c_s), (\widehat{\mathsf{u}}, \mathsf{b}_0, \mathsf{b}_1), q)$ is a promotable hybrid state if $\widehat{s}$ is* closed *and, an hybrid state otherwise, for all priority $q \in \mathsf{pr}$ such that $c_s < q \leq \min(\mathsf{rng}(\mathsf{r}_s \cup \mathsf{u}_s{}^{>c_s}))$, and $\mathsf{b}_0, \mathsf{b}_1 \in \mathbb{N}$.*

The function Und takes as input a strongly-maximal hybrid state and returns a pair of region and promotion functions $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$ that can be used to compose a new hybrid state. In particular, both $\widehat{\mathsf{r}}$ and $\widehat{\mathsf{u}}$ include the positions mapped above the priority p_s in the input state components r and u, respectively. The new hybrid state can be composed by swapping, in the input state, the old pair (r, u) with $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$ and the current priority p with the caller one c. Moreover, in case such new state is closed, then it is also promotable.

Lemma 7. *Given a maximal hybrid state $s \in \mathrm{St}_{\mathrm{M}}$, the function* **sol**$(s)$ *terminates and returns a pair $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \in \mathrm{Rg} \times \mathrm{Pf}$ of a region function $\widehat{\mathsf{r}}$ and a promotion function $\widehat{\mathsf{u}}$ such that: 1) $\mathsf{r}_s^{(>p_s)} \subseteq \widehat{\mathsf{r}}^{(>p_s)}$; 2) $\mathsf{u}_s^{(>p_s)} \subseteq \widehat{\mathsf{u}}^{(>p_s)}$; 3) $\widehat{s} = ((\widehat{\mathsf{r}}, c_s), (\widehat{\mathsf{u}}, \mathsf{b}_0, \mathsf{b}_1), q)$ is a promotable hybrid state if $\widehat{s}$ is* closed *and, an hybrid state otherwise, for all priorities $q \in \mathsf{pr}$ such that $c_s < q \leq \min(\mathsf{rng}(\mathsf{r}_s \cup \mathsf{u}_s{}^{>c_s}))$, and $\mathsf{b}_0, \mathsf{b}_1 \in \mathbb{N}$. Moreover, the procedure* **hsol**(s) *modifies in-place s into a strongly-maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{S}}$ such that $\widehat{s} \preccurlyeq s$.*

This lemma states that the function **sol** takes as input a maximal hybrid state and, when it terminates, it returns a pair of region and promotion functions $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$. In particular, both $\widehat{\mathsf{r}}$ and $\widehat{\mathsf{u}}$ include the positions mapped above the priority p_s in the input state components r and u, respectively. The new hybrid state can be composed by swapping, in the input state, the old pair (r, u) with $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$ and the current priority p with the caller one c. Moreover, in case such new state is closed, then it is also promotable. The procedure **hsol**, instead, modifies in-place a maximal hybrid state into a strongly-maximal hybrid state which is no greater than the original one *w.r.t.* $\prec$.

We can now proceed with the main lemma.

Lemma 8. *main Let $s \in \mathrm{St}_{\mathrm{M}}$ be a maximal hybrid state for a parity game $\mathcal{D}$, where b_{0s} and b_{1s} are positive. Assume* **sol** *is called on s and let $s' \triangleq ((\widehat{\mathsf{r}}, p'), (\widehat{\mathsf{u}}, b_{0s}, b_{1s}), c')$ be the hybrid state, for $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \triangleq$* **sol**$(s)$*, $p' \triangleq c_s$, and some $c' > c_s$. For $\alpha = \alpha_s$, the following holds:*

- *at the end of the call, H_s^{α} and $\mathrm{H}_s^{\overline{\alpha}}$ contain all small dominions of player α and size at most b_{α}, and small dominions of player $\overline{\alpha}$ and size at most $b_{\overline{\alpha}}$, respectively, in P_s;*
- *if $c_s \equiv_2 \alpha$ then $\mathrm{H}_s^{\overline{\alpha}} = \mathrm{H}_{s'}^{\overline{\alpha}}$; hence, $\mathrm{H}_{s'}^{\overline{\alpha}}$ preserves all global guarantees of $\mathrm{H}_s^{\overline{\alpha}}$;*
- *if $c_s \not\equiv_2 \alpha$ then $\mathrm{H}_{s'}^{\overline{\alpha}} = \mathrm{H}_s^{\alpha}$; hence, $\mathrm{H}_{s'}^{\overline{\alpha}}$ preserves the dual of all global guarantees of H_s^{α} (since the caller player c_s has dual parity w.r.t. α).*

Moreover, if **hsol** *is called on $s \in \mathrm{St}_{\mathrm{M}}$, then it modifies s into a strongly-maximal hybrid state with the following properties:*

- *at the end of the call, H_s^{α} and $\mathrm{H}_s^{\overline{\alpha}}$ contain all small dominions of player α and size at most b_{α}, and tiny dominions of player $\overline{\alpha}$ and size at most $\left\lfloor \frac{b_{\overline{\alpha}}}{2} \right\rfloor$, respectively, in P_s;*
- *L_s does not contain a small dominion of player $\overline{\alpha}$ of size at most $\left\lfloor \frac{b_{\overline{\alpha}}}{2} \right\rfloor$, and $\mathrm{H}_s^{\overline{\alpha}}$ is closed under attractor in P_s.*

Proof. We provide an inductive proof over the highest priority.

For the **induction basis** we consider the four cases in this order: (1) **sol** with highest priority $\bot$ (*i.e.* on the empty game); (2) **hsol** being called with highest priority 0; (3) **hsol** with highest priority $\bot$; and (4) **sol** being called with highest priority 0.

Case 1: When **sol** is called on the empty game the highest priority is $\bot$ and there is nothing to do (and due to Lines 1-2 **sol** does nothing).

Case 2: For games with maximal (and thus only) priority 0 in **hsol**, all states are in R_s, hence s is closed, and consequently, due to Lines 4 and 9, all states are promoted. The game is then empty (s is open), and by Lines 4 and 5, **sol** is called on the empty game. After that, Maximise is called (again on an empty set of positions). All global guarantees are retained due to Lemma 5, case 1, and Lemma 4. Moreover, as $\mathsf{dom}\big(\mathrm{r}_s^{\leq 0}\big) = \mathrm{u}_s^{-1}(0) = \emptyset$, the additional requirement for **hsol** holds.

Case 3: The case in which **hsol** is called on the empty game, is a sub-case of case 2 since s is open, and **sol** is called on the empty game.

Case 4: For games with maximal (and thus only) priority 0 in **sol**, all positions are promoted in the first call of **hsol** at Line 3 of Algorithm 5, similarly as shown for case 2. Then, s is empty and the successive call of **sol** at Line 5, of Maximise at Line 7, and of Und at Line 10 do nothing. The second call of **hsol** at Line 9 is skipped as s has not changed since Line 4.

Thus, the global guarantees are retained by the other base cases 1 and 2, and due to Lemmas 4 and 6.

We now implement the **induction step**, again first for **hsol**, and then for **sol**, using the results from **hsol**.

When executed, the call of **sol** at Line 5 of **hsol** provides, by induction hypothesis, an increase of $\mathrm{H}_s^{\overline{\alpha}}$, which now contains every small dominion of player $\overline{\alpha}$ of size at most $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$ in $\mathrm{L}_s \setminus \mathrm{R}_s$ from before the call, while retaining all global guarantees of H_s^{α} for which the bound is still b_{α_s}. When $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor = 0$, instead **sol** does nothing, trivially retaining the guarantees.

Thus, the global guarantees are preserved in every step either by induction hypothesis and Lemma 3, by the base case $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor = 0$, or by Lemmas 4 and 5 when functions Promote and Maximise are called.

Finally, to see that $\mathrm{H}_s^{\overline{\alpha}}$ is closed, we observe that it is closed after Maximise is called, due to Lemma 4. Moreover, in a generic iteration of **hsol**, if s is closed, it follows a promotion and then $s \prec \widehat{s}$ triggers a new iteration, otherwise when s is open the last operation executed is a call of Maximise. Hence, in both cases we have that $\mathrm{H}_s^{\overline{\alpha}}$ is closed when **hsol** ends.

So far we have shown that $\mathrm{H}_s^{\overline{\alpha}}$ contains all small $\overline{\alpha}$-dominions of size at most $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$ in $\mathrm{L}_s \setminus \mathrm{R}_s$. And since s has not changed during the call, it also contains the small dominions after the call, and on the time of return. To conclude this induction step we have to show that $\mathrm{H}_s^{\overline{\alpha}}$ contains also the small dominions in L_s. Do to this, we observe that R_s is an α-region with highest priority p_s, and that, by Lemma 2, for any non-empty $\overline{\alpha}$-dominion D' in L_s, there must be a non-empty sub-dominion $\mathrm{D}'' \subseteq \mathrm{D}'$, which is also a $\overline{\alpha}$-dominion in $\mathrm{L}_s \setminus \mathrm{R}_s$. As a consequence, since $\mathrm{L}_s \setminus \mathrm{R}_s$ does not contain a $\overline{\alpha}$-dominion D with size at most $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$ when **hsol** returns, we can conclude that neither does L_s.

We can now turn to the induction step for **sol**. We first check that the global guarantees are retained up to the point where Und is called, then we make the argument that **sol** will, for a $\overline{\alpha}$-dominion D in P_s of size $|\mathrm{D}| \leq b_{\overline{\alpha}_s}$, guarantee that $\mathrm{D} \subseteq \mathrm{H}_s^{\overline{\alpha}}$ holds. We will finally show that, under these conditions, Und will retain the global guarantees.

The check that the global guarantees are retained up to the point where Und is called is straightforward as we have seen that **hsol** (called at Lines 3 and 9) does by the induction step and Lemma 7, and both Maximise and Promote do retain the global guarantees by Lemmas 4 and 5, respectively.

We now argue that, when Und is called, $\mathrm{H}_s^{\overline{\alpha}}$ contains all $\overline{\alpha}$-dominions in P_s of size at most $b_{\overline{\alpha}_s}$. Let D be such a dominion.

We first observe that by the global guarantees $\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s$ and its α-attractor $\mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$, cannot intersect with D at any point. Thus, D is a dominion in $\mathrm{P}_s \setminus \mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$ at any point by Lemma 1.

After the first call of **hsol** (at Line 3), we have established by induction step that L_s does not contain a $\overline{\alpha}$-dominion of size at most $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$. Now two cases may arise: L_s does not intersect with D or it does. The first case entails that D must be contained in $\mathrm{H}_s^{\overline{\alpha}}$. In the second case, there is a set $\mathrm{D}' = \mathrm{D} \cap \mathrm{L}_s$ that, by Lemma 1, must be a $\overline{\alpha}$-dominion in L_s, as well as in $\mathrm{P}_s \setminus \mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$. By Lemma 2, we also know that D' has a sub-dominion $\mathrm{D}'' \subseteq \mathrm{D}'$ that does not intersect with R_s. Moreover, since L_s contains no smaller dominions than $\left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$, it follows that $|\mathrm{D}''| > \left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$. Thus, by induction hypothesis, after the call of **sol** from Line 5, $\mathrm{H}_s^{\overline{\alpha}}$ also contains D''. Note that in this case $s \prec \widehat{s}$ holds, and therefore **hsol** is called at Line 9. Again, by induction step, the global guarantees are retained, therefore also $\mathrm{D}' \setminus \mathrm{D}''$ is included in $\mathrm{H}_s^{\overline{\alpha}}$.

Let us assume for contradiction it is not. Then, after the return from **hsol**, there is a non empty set $\mathrm{E} = \mathrm{L}_s \cap (\mathrm{D}' \setminus \mathrm{D}'')$ that still belongs to the current subgame. Clearly $\mathrm{D} \supseteq \mathrm{D}' \supseteq \mathrm{D}' \setminus \mathrm{D}''$, and since $|\mathrm{D}| \leq b_{\overline{\alpha}_s}$ and $|\mathrm{D}''| > \left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$, it easily follows that $|\mathrm{E}| \leq \left\lfloor \frac{b_{\overline{\alpha}_s}}{2} \right\rfloor$. Moreover, by the global guarantees, E cannot intersect with $\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s$ or with its attractor $\mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$. Hence, E is contained in $\mathrm{P}_s \setminus \mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$, and $\mathrm{H}_s^{\overline{\alpha}}$ (that now includes D'') is closed under $\overline{\alpha}$-attractor in P_s. Thus, by Lemma 2, E must be a $\overline{\alpha}$-dominion in L_s, as well as in $\mathrm{P}_s \setminus \mathsf{atr}^{\alpha}\big(\mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s\big)$. However, due to the inductive hypothesis, after the call of **hsol** at Line 9 we have that L_s cannot contain such a small dominion for player $\overline{\alpha}$, contradicting the assumption that E is not empty.

We have now established that the global guarantees hold when Und is called, as $H_s^{\overline{\alpha}}$ contains all $\overline{\alpha}$-dominions of size at most $b_{\overline{\alpha}_s}$ in P_s. This entails that $H_s^{\alpha} \cup L_s$ contains all α-dominions of size at most b_{α_s} in P_s, when Und is called.

Und then moves the respective set, U_s (or L_s), to $u_s^{-1}(c_s)$ of the caller priority c_s, such that $H_{s'}^{\overline{\alpha}}$ is equal to $H_s^{\overline{\alpha}}$ (or to $H_s^{\alpha} \cup L_s$) when $c_s \equiv_2 p_s$ (or $c_s \not\equiv_2 p_s$, respectively). Thus, due to Lemma 6, Und retains the global guarantees.

This completes the induction step for **sol** and the proof. □

Correctness is then the special case when **sol** is called with $c_s = \top_{\overline{\alpha}}$, where $\alpha \triangleq \mathrm{pr}(\mathcal{D}) \bmod 2$, and full precision, *i.e.* $b_\alpha = |\mathcal{D}|$, for all $\alpha \in \mathbb{B}$.

Theorem 2. *When* **sol** *is called with the initial state s_I, i.e. with $c_s = \top_{\overline{\alpha_I}}$, where $\alpha_I \triangleq \mathrm{pr}(\mathcal{D}) \bmod 2$, and full precision $b_\alpha = |\mathcal{D}|$, for all $\alpha \in \mathbb{B}$, then, after* **sol** *returns, $r^{-1}(\top_{\alpha_I}) \cup u^{-1}(\top_{\overline{\alpha_I}})$ is the winning region of player α_I.* □

The theorem immediately follows from the previous lemma since the initial state has full precision and, therefore, the guarantees are not bounded. The assignment of the winning regions is due to the fact that $r^{-1}(\top_0)$ and $r^{-1}(\top_1)$ contain dominions of the respective players and are closed under attractor by our global guarantees. It is also clear that $u^{-1}(\top_{\overline{\alpha_I}})$ can only be filled in the final call of Und, such that $r^{-1}(\top_{\alpha_I}) \cup u^{-1}(\top_{\overline{\alpha_I}})$ contains all dominions (as the bound does not exclude any) of Player α_I, but does not intersect with any dominion (as the bound again does not include any) dominion of Player $\overline{\alpha_I}$. Note that the wining region of Player 0 is simply the complement of the winning region of Player 1.

It is interesting to note an algorithmic difference between the parts of the winning regions in the dominions $r^{-1}(\top_0)$ and $r^{-1}(\top_1)$, and the rest of the winning regions of both players. The two sets are computed constructively, and winning strategies are simply the contribution of attractor strategies. This is not the case for the remainder of the winning regions, as their calculation is not constructive.

As a hybrid between Priority Promotion and Parys' approach, the algorithm retains the quasi-polynomial bound of Parys and the practical efficiency of Priority Promotion from [26–28]. Our argument is exactly the same as Parys' (end of Section 4 of [59]): we use 2 parameters, h for the number of priorities, and $l = 2\lfloor \log_2(n) \rfloor + 1$ for the bounds, where n is the number of positions. We estimate the number of times **sol** is called, excluding the trivial calls that return immediately (at Line 2) because we run out of priorities or bounds, by $R(h, l)$. If $h = 0$, then we have run out of priorities ($p = \bot$), and $R(h, l) = 0$. If $l = 0$, then we have run out of bounds ($b_0 = 0$ or $b_1 = 0$, with the other value being 1). As argued in Section 4 of [59], we can estimate $R(h, l) \leq 2n^l \binom{h+l}{l} - 1$. Therefore, since the cost of all operations is linear in the size of the game, and since l is logarithmic in the number of positions, this provides a quasi-polynomial running time.

7. Experimental evaluation

The practical effectiveness of the algorithms presented here, namely RPP and HPP, has been assessed by means of an extensive experimentation on both concrete and synthetic benchmarks. The algorithms have been incorporated in Oink [64], a tool written in C++ that collects implementations of several parity game solvers proposed in the literature, including some of the known quasi-polynomial ones. We shall compare solution times against the quasi-polynomial solvers SSPM [51], QPT [50], and the improved version of Parys' algorithm PAR [58], as well as the original version of the best exponential solver classes, namely the recursive algorithm ZLK from [30] and the original priority promotion PP [27], whose superiority in practical contexts is widely acknowledged[1] (see *e.g.*, [68,64]).

The results give a simple argument for why and when to use this algorithm.[2] The first question is why one should use a QP algorithm. The answer to that question is quite simple: it is not hard to produce pathological cases for exponential time solvers. For complex games, it is well known that even the most efficient solvers in practice, *i.e.* ZLK and PP [42,24,69,44], take exponential time, while HPP has a quasi-polynomial worst case complexity. To show this behaviour, we have evaluated these two solvers, PP and ZLK, with HPP and the improved version of Parys' algorithm PAR, a quasi polynomial version of ZLK, on the worst case family developed for the approaches based on quasi-dominions [24]. The results are reported in Table 1. Clearly the complex infrastructure required by the HPP can pay off in terms of running time, while PAR does not outperform ZLK on these small examples.

The reason for why most QP algorithms should not be used in practice becomes evident when the algorithms are run on Keiren's family of (for current solvers) simple benchmarks [70]. This set of benchmarks comprises a number of concrete verification problems, ranging from model-checking to equivalence-checking and decision problems for different temporal logics. They can be divided in the following four categories.

Model-checking benchmarks. The first group contains 313 games, with size up to in the order of magnitude of 10^7 positions. It includes a number of different verification problems. A first set contains encodings of a variety of communication protocols from [71–74]: the alternating bit protocol, the positive acknowledgement with retransmission protocol, the bounded

[1] Variations of Zielonka's algorithm [65] as well as of Priority Promotion approaches [26–28] (including Tangle Learning [66] and Justification [67] based approaches), who share the same basic data structure and promotion principles, are available. Their performance on benchmarks does not vary significantly. We therefore went with the original Parity Promotion approach to compare the principle performance.

[2] Experiments were carried out on a 64-bit 3.9 GHz Intel quad-core machine, with i5-6600K processor and 16 GB of RAM, running Ubuntu 18.04 with Linux kernel version 4.15.0. Oink was compiled with gcc 7.4.

Table 1
Solution times in seconds on the worst case family [24].

Worst Case		PP		PAR		ZLK		HPP	
Index	Nodes	Time	Iterations	Time	Iterations	Time	Iterations	Time	Iterations
10	88	0.00	11267	0.00	12750	0.00	1295	0.00	69
15	170	0.18	524294	0.21	502439	0.00	10175	0.00	177
20	278	7.70	22020104	4.54	9561994	0.03	133631	0.00	339
25	410	-	-	51.13	82520213	0.17	796671	0.00	879
30	568	-	-	-	-	1.99	8863743	0.00	1217
35	750	-	-	-	-	11.25	47120383	0.01	2125
40	958	-	-	-	-	-	-	0.02	2952

Table 2
Solution times in seconds on Keiren's benchmarks (1012 games).

		Exponential			Quasi Polynomial			
Benchmarks		ZLK	PP	RPP	SSPM	QPT	PAR	**HPP**
Model-Ch.	Tot. Time	27.16	20.12	53.66	>849.01	>1259.84	42.38	**64.21**
	Avg. Time	0.08	0.06	0.17	>2.71	>4.02	0.13	**0.2**
	TimeOut	0%	0%	0%	22%	36.4%	0%	**0%**
Equiv. Ch.	Tot. Time	202.95	137.92	242.32	>2681.33	>3139.85	208.3	**280.81**
	Avg. Time	0.94	0.63	1.12	>12.41	>14.53	0.93	**1.3**
	TimeOut	0%	0%	0%	28.2%	27.3%	0%	**0%**
Decision Prb.	Tot. Time	13.20	11.75	13.27	>360.8	>853.64	66.85	**14.27**
	Avg. Time	0.07	0.06	0.07	>1.87	>4.44	0.35	**0.07**
	TimeOut	0%	0%	0%	11%	26.5%	0%	**0%**
PGSolver	Tot. Time	1.54	2.21	2.89	>3615.22	>4069.12	8.62	**4.04**
	Avg. Time	0.005	0.007	0.009	>12.42	>13.98	0.03	**0.01**
	TimeOut	0%	0%	0%	78%	92.4%	0%	**0%**

retransmission protocol, and the sliding window protocols. The protocols are parameterised with the number of messages to send and, when applicable, the window size. The set also contains verification problems for the cache coherence protocol of [75] and the wait-free handshake register of [76], as well as the classic elevator and towers-of-Hanoi benchmarks from [77]. The verification tasks under analysis cover fairness, liveness and safety properties. A second set, instead, contains encodings of two-player board games, such as Clobber, Domineering, Hex, Othello, and Snake, all parameterised by their board size. Here, the existence of a winning strategy for the game is the property considered. The encoding into parity games results in games with very few priorities: up to 4 in some cases.

Equivalence checking benchmarks. This group contains 216 games encoding equivalence tests between processes. The verification problems test various forms of process equivalences, such as strong, weak and branching bisimulation, as well as branching simulation. Most of the processes are the ones already considered in the model-checking benchmarks. The encoding into parity games results in games with at most two priorities, hence the only relevant measure of difficulty is the size, again reaching $O(10^7)$ positions for the bigger instances.

Decision problem benchmarks. The third group contains encodings of satisfiability and validity problems for formulae of various temporal logics: LTL, CTL, CTL*, PDL and the μ-CALCULUS, and comprises 192 games. The maximal size of a benchmark is around $3 \cdot 10^6$ positions. The parity games encoding has been obtained with the tool MLSolver [78]. The situation here is more interesting, since these concrete problems feature a higher number of priority, up to 20 in few cases. Hence, unlike the previous two groups, these benchmarks allow us to stress a bit more the scalability of the solution algorithms *w.r.t.* the increase in priorities.

PGSolver. This group contains 291 synthetic benchmarks, corresponding to known families of hard cases for specific solvers and randomly generated ones. The sizes and number of priorities vary significantly, depending on the specific class of games.

Table 2 reports the results of the experiments for all the solvers considered in the analysis, divided by class of benchmarks.[3] For each solver, the total completion time, the average time per benchmark and the percentage of timed-out executions are given. We set a timeout of 10 seconds for all the benchmarks, except for the equivalence-check class, for which 40 seconds are used instead. As expected, the exponential solvers perform better on all the classes, with PP taking the lead most of the time. SSPM and QPT both perform quite poorly, between two and three orders of magnitude worse than the other solvers, and do not seem to scale beyond the simplest instances, as also evidenced by the high number of

[3] The benchmarks were run by issuing the following OINK commands: oink –no-single –no-loops –no-wcwc *GameName SolverName*; where solvers are: ZLK, NPP, RPP, SSPM, QPT, ZLKQ.

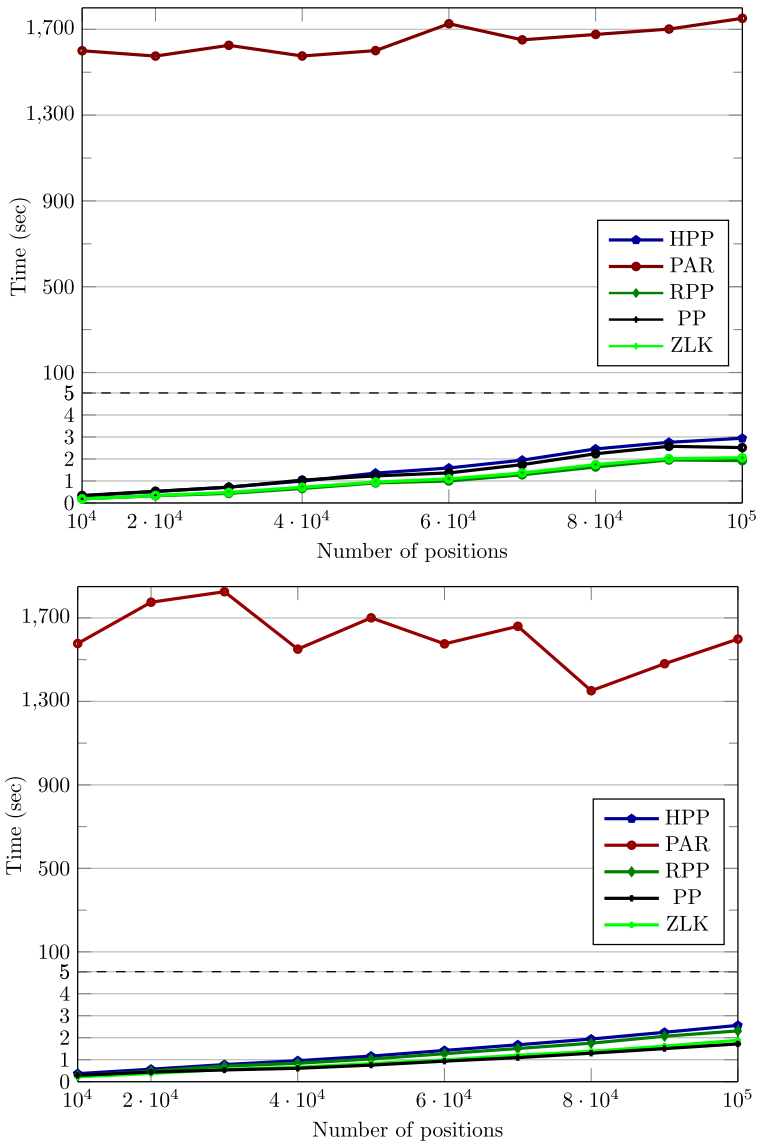

Fig. 5. Results on random games with linear (on the left) and fixed (on the right) number priorities.

timeouts. Both PAR and HPP, instead, perform relatively well in all the benchmarks, being able to solve all the instances without incurring any timeouts and maintaining a short distance from the exponential solvers performance-wise. PAR has a slight edge over HPP on the model-checking and equivalence checking problems, both of which feature a very low number of priorities, though the time advantage on average is typically negligible. On the other hand, when the number of priorities increases, like in the decision problems, the situation reverses and HPP takes the lead over PAR and practically matches the performance of the exponential solvers. We can observe the same behaviour also for the non-bounded recursive version RPP, whose performance relies less on the number of priorities, so that its advantage increases with a rising number of priorities. This seems to suggest that HPP may scale better *w.r.t.* the number of priorities in the games. To further investigate this behaviour we decided to perform additional experiments.

We then tried to assess scalability *w.r.t.* the number of positions and priorities, so as to evaluate how sensitive the solvers are to variations of those two parameters. To this end, we set up two types of synthetic benchmarks. The first kind of benchmarks keeps the number of priorities fixed and only increases the size of the underlying graph, while the second one maintains a linear relation between positions and priorities. Here we drop both SSPM and QPT, since they could not solve any of these benchmarks.

Fig. 5 reports the solution times of the quasi polynomial solvers on 10 clusters, each composed of 100 randomly generated games, of increasing size varying from 10^4 to 10^5. Each point corresponds to the total time to solve all the games in

the cluster.[4] On the top graph the number of priorities grows linearly with the positions, *i.e.*, equal to $\frac{n}{4}$, with n the number of positions. On the bottom graph, instead, all the games have 500 priorities. In both cases, the timeout was set to 25 seconds. In these experiments, HPP and PAR are tested together with the exponential solvers. The results seem to confirm what we already observed with the decision procedure benchmarks of the previous subsection. HPP definitely scales very well *w.r.t.* the number of priorities, as opposed to PAR, which is very sensitive to this parameter and starts hitting the timeout already on the smallest instances. HPP, indeed, behaves very much like the exponential solvers, none of which seems to be particularly sensitive to this parameter in practice, despite requiring time exponential in the number of priorities in the worst case.

What seems to emerge from the experimental analysis is that HPP behaves quite nicely in practice, competing with the leading exponential solution algorithms. The algorithmic overhead that guarantees its quasi-polynomial upper bound does not seem to impact the performance in any meaningful way. This is in contrast to what happens for all the other quasi-polynomial solvers, which do *not* scale with the number of positions and/or the number of priorities. This bodes well for the applicability of the hybrid approach in more challenging practical contexts, such as deciding temporal logic properties or solving reactive synthesis problems, where the number of priorities is typically higher.

Thus, HPP should be used when the game is hard; for this, it should have *some* structure (as opposed to be essentially random) as well as a high number of priorities, as the used advanced data structure only kicks in when the number of priorities is higher than $\log(n)$.

The excellent performance of the basic interplay between the two parts of the data structure invites exploring the limits of this approach. For example, one could refine the interplay between these parts in our algorithm, by extracting the guarantees on the bounds provided upon return instead of the guarantees required when a call is made.

As a final remark regarding the computation of the strategy, a very relevant question in reactive synthesis applications, we have observed that, in all the experiments we conducted, the final undetermined region is always empty. This means that, in practice, the winning strategies of the two players are always complete.

8. Discussion

We have combined two generalisations of the classic recursive algorithm: the quasi-polynomial recursion scheme of Parys, which relies on the local spread of imperfect guarantees, and a Priority Promotion scheme, which relies on identifying and realising the global potential of perfect guarantees. That these improvements can be synthesised bodes well, as it promises to perfectly join the advantages of both schemes, and the first experimental data collected suggests that the algorithm lives up to this promise.

CRediT authorship contribution statement

Massimo Benerecetti: Conceptualization, Formal analysis, Validation, Writing – original draft, Writing – review & editing. **Daniele Dell'Erba:** Conceptualization, Formal analysis, Software, Validation, Writing – original draft, Writing – review & editing. **Fabio Mogavero:** Conceptualization, Formal analysis, Validation, Writing – original draft, Writing – review & editing. **Sven Schewe:** Conceptualization, Formal analysis, Validation, Writing – original draft, Writing – review & editing. **Dominik Wojtczak:** Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101032464. F. Mogavero acknowledges a partial support by GNCS 2020 project "Ragionamento Strategico e Sintesi Automatica di Sistemi Multi-Agente". M. Benerecetti and F. Mogavero acknowledge a partial support by GNCS 2024 project "Certificazione, Monitoraggio, ed Interpretabilita' in Sistemi di Intelligenza Artificiale". The work was supported by EPSRC grants EP/P020909/1 and EP/X017796/1.

[4] The instances were generated by issuing the following OINK command for top graph games: rngame n n/4 2 10; bottom graph games: rngame n 500 2 10.

The authors would like to thank the anonymous reviewers for the insightful comments and suggestions (in particular regarding the experimental analysis on the strategy coverage) that helped to considerably improve the earlier version of the manuscript.

Appendix A. Properties of the auxiliary functions for Section 6

Lemma 9. *Given a strongly-maximal hybrid state $s \in \mathrm{St}_{\mathrm{S}}$, the functions* NextPr$(s)$ *and* Half(s) *return a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \prec s$.*

Proof. Let $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$ be a strongly-maximal hybrid state. For function NextPr, let $\widehat{s} = \mathsf{NextPr}(s)$ be the result obtained by calling the function NextPr on s. Due to Line 2 of Algorithm NextPr, $\widehat{s} = ((\mathsf{r}_s, q), (\mathsf{u}_s, b_{0s}, b_{1s}), p_s)$ where $q = \max(\mathrm{rng}(\mathsf{r}_s^{(<p_s)}))$ by Line 1. Consequently, trivially holds Condition 1a and 1d of Definition 7 and that r_s is a region function, u_s is a promotion function, p_s is a priority, and b_α for $\alpha \in \mathbb{B}$ are two integers. Moreover, clearly $q \in [\bot, p_s[$ and, therefore, also q is a priority and both $\{v \in \mathsf{dom}(\mathsf{u}_s) \mid \mathsf{u}_s(v) < p_s\}$ and $\mathsf{u}_s^{-1}(\top_\alpha)$, where $\alpha \triangleq \mathsf{pr}(\ominus) \bmod 2$, are equal to $\emptyset$ as required by Condition 1b of the same definition. In addition, since $\mathrm{H}_s^\beta = \mathrm{H}_{\widehat{s}}^\beta \cup \mathsf{r}^{-1}(q)$ and $\mathrm{H}_{\widehat{s}}^{\overline{\beta}} = \mathrm{H}_s^{\overline{\beta}}$, with $\beta = q \bmod 2$, it also follows Condition 1c. Finally, to prove that $\widehat{s}$ is maximal it suffices to observe that $\mathrm{H}_{\widehat{s}}^\alpha = \mathrm{H}_s^\alpha \cup \mathrm{R}_s$ and $\mathrm{H}_{\widehat{s}}^{\overline{\alpha}} = \mathrm{H}_s^{\overline{\alpha}}$. Consequently, since s is strongly-maximal it holds that R_s is α-maximal *w.r.t.* L_s and, therefore, $\mathrm{H}_{\widehat{s}}^\alpha$ is α-maximal as well. To conclude, if $\mathrm{R}_s \neq \emptyset$ trivially we have that $\mathrm{L}_{\widehat{s}} \subset \mathrm{L}_s$, otherwise we have that $\mathrm{L}_{\widehat{s}} = \mathrm{L}_s$ and $p_{\widehat{s}} = q < p_s$. In both cases we can have that $\widehat{s} \prec s$.

We can now focus on Half function. Let us consider $\widehat{s}$ as the argument of function NextPr at Line 2 of Algorithm Half. Due to Line 1, we have that $\widehat{s} = ((\mathsf{r}, p), (\mathsf{u}, \widehat{b_0}, \widehat{b_1}), c)$, where $\widehat{b_0} = \lfloor \frac{b_{0s}}{1+\alpha} \rfloor$ and $\widehat{b_1} = \lfloor \frac{b_{1s}}{2-\alpha} \rfloor$. It is easy to observe, by the fact that s is a strongly-maximal hybrid state and that both $\widehat{b_0}$ and $\widehat{b_1}$ are integers, that $\widehat{s}$ is a strongly-maximal hybrid state as well. Hence, Half returns NextPr$(\widehat{s})$ that in turn returns a new maximal hybrid state $s' \in \mathrm{St}_{\mathrm{M}}$ for which $s' \prec \widehat{s}$. □

Lemma 10. *Given a hybrid state $s \in \mathrm{St}$, the procedure* Maximise(s) *modifies in-place s into a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \preccurlyeq s$.*

Proof. Let $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$ be an hybrid state and $\widehat{s} \triangleq ((\widehat{\mathsf{r}}, \widehat{p}), (\widehat{\mathsf{u}}, \widehat{b_0}, \widehat{b_1}), \widehat{c}) = \mathsf{Maximise}(s)$ the modified state obtained by computing the function Maximise on s. It is easy to see that $\widehat{p} = p$, $\widehat{c} = c$, and $\widehat{b_\alpha} = b_\alpha$ for $\alpha \in \mathbb{B}$. Moreover, due to Line 3 of Algorithm Maximise, it holds that $\mathrm{X} \subseteq \mathrm{L}_s \cup \mathrm{U}_s \subseteq \{v \in \mathsf{dom}(\mathsf{pr}) \mid \mathsf{pr}(v) \leq p_s\}$ while, due to Line 4, we have that $q \geq p$. As a consequence, by Lines 6 and 8, $\widehat{\mathsf{r}}$ is a promotion function. Furthermore, from the observation that in case $q = \top_\beta$, with $\beta \in \mathbb{B}$, it necessarily holds that $\alpha \equiv_2 \beta$, we can also conclude that, by Lines 7 and 9, $\widehat{\mathsf{u}}$ is a promotion function as well. To prove that $\widehat{\mathsf{r}}$ is a region function, let us consider again the case in which $q = \top_\beta$ with $\beta \in \mathbb{B}$. Now, since $\alpha \equiv_2 \beta$, the set X corresponds to the β-attractor of $\mathrm{L}_s \cup \mathrm{U}_s$ by Line 3. Once X is merged to $\mathsf{r}^{-1}(\top_\beta)$, by Line 6, it is obvious that $\mathsf{r}^{-1}(\top_\beta)$ is still a β-dominion due to the fact that the β-attractor of a β-dominion is a β-dominion as well. We can now consider $q < \top_\beta$ for any $\beta \in \mathbb{B}$. Let us first assume that at Line 2 $\alpha = \alpha_s$. Two cases may arise: $q \equiv_2 \alpha$ or $q \not\equiv_2 \alpha$. In the first case, due to Lines 5-6, we have that $\Delta_{\widehat{\mathsf{r}}}^\alpha = \Delta_{\mathsf{r}}^\alpha \cup \mathrm{X}$ which is clearly a quasi α-dominion since X corresponds to the α-attractor of L_s, while $\Delta_{\widehat{\mathsf{r}}}^{\overline{\alpha}} = \Delta_{\mathsf{r}}^{\overline{\alpha}} \setminus \mathrm{X}$ is a quasi $\overline{\alpha}$-dominion. Hence, $\widehat{\mathsf{r}}$ is a quasi dominion function. Finally, clearly X does not belong to $\mathsf{esc}^{\overline{\alpha}}(\mathrm{H}^\alpha)$, since Player α has a strategy to attract every position $v \in \mathrm{X}$ to $\mathsf{r}^{-1}(q)$, from which the opponent can only escape through positions having priority $q' \geq q$ and congruent to the parity of q. Summing up, the resulting $\widehat{\mathsf{r}}$ is a region function and due to the fact that the priority function pr is always a region function, we can conclude that also after Line 11 $\widehat{\mathsf{r}}$ is a region function. On the other hand, if $q \not\equiv_2 \alpha$, we can easily conclude that $\widehat{\mathsf{r}}$ is a region function due to Lines 5, 8, and 11. When, instead, at Line 2 $\alpha \neq \alpha_s$ and the priority selected at Line 4 has the same parity as α, the proof that $\widehat{\mathsf{r}}$ is a region function is equivalent to the one provided above for the case in which $q \not\equiv_2 \alpha$ and $\alpha = \alpha_s$. Similarly, when $q \not\equiv_2 \alpha$ the proof corresponds to the one provided for the case in which $q \equiv_2 \alpha$ and $\alpha = \alpha_s$. To prove Condition 1a of Definition 7 it suffices to observe that whenever X is merged to r (resp. u), it is removed from u (resp. u) due to Lines 6-7 and 8-9, while the operation at Line 11 does not change $\mathsf{dom}(\mathsf{r})$, since R_s is already stored in r. Moreover, due to Line 4, q refers to a priority in the domain of r or u, hence also Condition 1d holds. Condition 1b, instead, follows from the fact that $q \geq p$ and $q = \top_\beta$ with $\beta \in \mathbb{B}$ only if $\alpha \equiv_2 \beta$. Therefore, it holds that $\widehat{\mathsf{u}}^{-1}(\top_\alpha) = \mathsf{u}^{-1}(\top_\alpha)$, where $\alpha \triangleq \mathsf{pr}(\ominus) \bmod 2$, and $\{v \in \mathsf{dom}(\widehat{\mathsf{u}}) \mid \widehat{\mathsf{u}}(v) < p\} = \{v \in \mathsf{dom}(\mathsf{u}) \mid \mathsf{u}(v) < p\}$ which are all empty. It remains to prove Condition 1c. Since s is a hybrid state, it holds that U_s and every $v \in \Delta_{\mathsf{u}}^{\overline{\alpha}}$ cannot be directly attracted by R_s and more in general by L_s, due to Condition 1c. In addition, by Lines 5 and 9, the domain of u only increases when the priority selected at Line 4 has parity $\overline{\alpha}$. However, the positions of X merged in this case are not part of $\mathsf{esc}^{\overline{\alpha}}(\mathrm{H}_{\widehat{s}}^{\overline{\alpha}})$ since X has been attracted to $\mathsf{dom}(\mathsf{u}_s) \cap \mathrm{H}_s^{\overline{\alpha}}$ and, therefore, Player $\overline{\alpha}$ has a strategy to reach this set from X. The maximality of $\widehat{s}$ is a trivial consequence of Line 3, together with the observation that the operation at Line 11 cannot affect the maximality of the state. Finally, since $\mathsf{dom}(\widehat{\mathsf{u}}) \supseteq \mathsf{dom}(\mathsf{u})$ and $\mathsf{dom}(\widehat{\mathsf{r}}) \supseteq \mathsf{dom}(\mathsf{r})$, while $\widehat{p} = p$, we can also conclude that $\widehat{s} \preccurlyeq s$. □

Lemma 11. *Given a promotable hybrid state $s \in \mathrm{St}_{\mathrm{P}}$, the procedure* Promote$(s)$ *modifies in-place s into a maximal hybrid state $\widehat{s} \in \mathrm{St}_{\mathrm{M}}$ such that $\widehat{s} \prec s$.*

Proof. Let $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$ be a promotable hybrid state and $\widehat{s} \triangleq ((\widehat{\mathsf{r}}, \widehat{p}), (\widehat{\mathsf{u}}, \widehat{b_0}, \widehat{b_1}), \widehat{c}) = \mathsf{Promote}(s)$ the modified state obtained by computing the function Promote on s. It is easy to see that $\widehat{p} = p$, $\widehat{c} = c$, and $\widehat{b_\alpha} = b_\alpha$ for $\alpha \in \mathbb{B}$. At Line 1 of Algorithm Promote the two best escape priorities $(p_\mathsf{r}, p_\mathsf{u})$ from R_s to r and u are computed by the function bep. Due to the definition of bep and the fact that R_s is an α-dominion in $\mathcal{D}_s$, it follows that both p_r and p_u are priorities higher than p. Moreover, by the same definition of bep and the fact that s is promotable, it also holds that $p_\mathsf{r} \equiv_2 p$, while $p_\mathsf{u} \not\equiv_2 p$ if $p_\mathsf{u} < \top_{\alpha_s}$. Now, two cases may arise: $p_\mathsf{r} \leq p_\mathsf{u}$ or $p_\mathsf{r} > p_\mathsf{u}$. In the first case, by Line 2 and 3, we have that $\widehat{\mathsf{u}} = \mathsf{u}$ and $\widehat{\mathsf{r}} = \mathsf{r}[\mathsf{R}_s \mapsto p_\mathsf{r}]$. Hence, $\widehat{\mathsf{u}}$ is a promotion function and Condition 1b of Definition 7 is satisfied. Moreover, since $\mathsf{R}_s \subseteq \mathsf{r}^{(\leq p)}$, we have that $\mathsf{dom}(\widehat{\mathsf{r}}) = \mathsf{dom}(\mathsf{r})$ and, therefore, also the two requirements of Condition 1a are guaranteed. By the definition of bep, p_r belongs to the domain of r or is equal to the pseudo priority $\top$. In both cases Condition 1d still holds. Finally, since $p_\mathsf{r} \equiv_2 p$, it follows that $\mathsf{H}^\beta_{\widehat{s}} = \mathsf{H}^\beta_s$ for any $\beta \in \mathbb{B}$. Hence, Condition 1c is satisfied. At this point, it remains to prove that $\widehat{\mathsf{r}}$ is a region function. Now, since $\mathsf{R}_s \subseteq \mathsf{r}^{(\leq p)}$ and $p_\mathsf{r} > p$, $\widehat{\mathsf{r}}$ is clearly a promotion function. In addition, it is also a quasi dominion function as a consequence of the fact that $\mathsf{H}^\beta_{\widehat{s}} = \mathsf{H}^\beta_s$ for any $\beta \in \mathbb{B}$. Moreover, being r a region function, it holds that $\mathsf{pr}(v) \geq p$ and $\mathsf{pr}(v) \equiv_2 \alpha$ for each position v in $\mathsf{esc}^{\overline{\alpha}}(\Delta^{\alpha,p}_\mathsf{r})$. However, due to Line 1 we have that $\mathsf{bep}^{\overline{\alpha}}_{\mathcal{D}}(\mathsf{R}_s, \mathsf{r}_s) = p_\mathsf{r}$, which implies that $\mathsf{pr}(v) \geq p_\mathsf{r}$ and $\mathsf{pr}(v) \equiv_2 \alpha$ for each position v in $\mathsf{esc}^{\overline{\alpha}}(\Delta^{\alpha,p}_\mathsf{r})$. Hence, $\widehat{\mathsf{r}}$ is a region function. Finally, since s is promotable, it follows that $\mathsf{H}^\alpha_{\widehat{s}} = \mathsf{H}^\alpha_s \cup \mathsf{R}_s$ is α-maximal, while $\mathsf{H}^{\overline{\alpha}}_{\widehat{s}} = \mathsf{H}^{\overline{\alpha}}_s$ is $\overline{\alpha}$-maximal and, therefore, $\widehat{s}$ is maximal. To conclude, being s promotable, it necessarily holds that $\mathsf{R}_s \neq \emptyset$, which implies that $\mathsf{L}_{\widehat{s}} \subset \mathsf{L}_s$. Hence, $\widehat{s} \prec s$. Let us now consider the case in which $p_\mathsf{r} > p_\mathsf{u}$. Due to Line 2 and 4 of Algorithm Promote we have that $\widehat{\mathsf{r}} = \mathsf{r} \setminus \mathsf{R}_s$ and $\widehat{\mathsf{u}} = \mathsf{u}[\mathsf{R}_s \mapsto p_\mathsf{u}]$. Conditions 1a and 1b follow from the fact that $\top_{\alpha_s} > p_\mathsf{u} > p$ and $\mathsf{R}_s \subseteq \mathsf{r}^{(\leq p)}$, which also imply that $\widehat{\mathsf{u}}$ is a promotion function. In addition, since $\mathsf{R}_s \triangleq \mathsf{r}^{-1}(p)$, it holds that $\Delta^{\alpha,p}_{\widehat{\mathsf{r}}} = \Delta^{\alpha,p}_\mathsf{r} \setminus \mathsf{R}_s$, hence $\widehat{\mathsf{r}}$ is a region function. By the definition of bep, p_r belongs to the domain of u, hence Condition 1d still holds. To prove Condition 1c, instead, let us observe that R_s is an α-dominion in $\mathcal{D}_s$ and, consequently, $\mathsf{esc}^\alpha(\mathsf{R}_s) \subseteq \mathsf{H}^\alpha_s$. Moreover, we have that $p_\mathsf{u} \not\equiv_2 p$, hence, as a consequence of the fact that $\widehat{\mathsf{u}} = \mathsf{u}[\mathsf{R}_s \mapsto p_\mathsf{u}]$, it holds that $\mathsf{R}_s \in \Delta^{\overline{\alpha},p}_{\widehat{\mathsf{u}}} \subseteq \mathsf{H}^{\overline{\alpha}}_{\widehat{s}}$. In addition, it suffices to observe that $\{v \in \mathsf{dom}(\widehat{\mathsf{r}}) \mid \widehat{\mathsf{r}}(v) > p\} = \{v \in \mathsf{dom}(\mathsf{r}) \mid \mathsf{r}(v) > p\}$ and $\{v \in \mathsf{dom}(\widehat{\mathsf{u}}) \mid \widehat{\mathsf{u}}(v) \geq p\} = \{v \in \mathsf{dom}(\mathsf{u}) \mid \mathsf{u}(v) \geq p\} \cup \mathsf{R}_s$, with $\mathsf{R}_s \neq \emptyset$, to conclude that $\widehat{s} \prec s$. Finally, similarly to the previous case, it can be proved that both $\mathsf{H}^\alpha_{\widehat{s}} = \mathsf{H}^\alpha_s \cup \mathsf{R}_s$ is α-maximal, while $\mathsf{H}^{\overline{\alpha}}_{\widehat{s}} = \mathsf{H}^{\overline{\alpha}}_s$ is $\overline{\alpha}$-maximal, as a direct consequence of the fact that s is promotable. □

Lemma 12. *Given a strongly-maximal hybrid state $s \in \mathsf{St}_\mathsf{S}$, the function* Und$(s)$ *returns a pair $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \in \mathsf{Rg} \times \mathsf{Pf}$ of a region function $\widehat{\mathsf{r}}$ and a promotion function $\widehat{\mathsf{u}}$ such that: 1) $\mathsf{r}_s^{(>p_s)} \subseteq \widehat{\mathsf{r}}^{(>p_s)}$; 2) $\mathsf{u}_s^{(>p_s)} \subseteq \widehat{\mathsf{u}}^{(>p_s)}$; 3) $\widehat{s} = ((\widehat{\mathsf{r}}, c_s), (\widehat{\mathsf{u}}, \mathsf{b}_0, \mathsf{b}_1), q))$ is a promotable hybrid state if $\widehat{s}$ is* closed *and, an hybrid state otherwise, for all priority $q \in \mathsf{pr}$ such that $c_s < q \leq \mathsf{min}(\mathsf{rng}(\mathsf{r}_s \cup \mathsf{u}_s{}^{>c_s}))$, and $\mathsf{b}_0, \mathsf{b}_1 \in \mathbb{N}$.*

Proof. Let $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$ be a strongly-maximal hybrid state and $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$ the result obtained by computing the function Und on s. Moreover, let $\widehat{s} = ((\widehat{\mathsf{r}}, c), (\widehat{\mathsf{u}}, \mathsf{b}_0, \mathsf{b}_1), q)$ the composition of state s and the pair $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$. Two cases may arise: $c \equiv_2 \alpha$ or $c \equiv_2 \overline{\alpha}_s$. In the first case, by Lines 1-2 of Algorithm Und, we have that $\widehat{\mathsf{r}} = \mathsf{r}$ and $\widehat{\mathsf{u}} = \mathsf{u}[\mathsf{U}_s \mapsto c]$, where $\mathsf{U} \triangleq \mathsf{u}^{-1}(p)$. By the fact that the priorities of the states are strictly decreasing we have that $c > p$. Moreover, it holds that $\mathsf{dom}(\widehat{\mathsf{u}}) = \mathsf{dom}(\mathsf{u})$ and, therefore, $\widehat{\mathsf{r}}$ is a region function, $\widehat{\mathsf{u}}$ is a promotion function, both Points 1) and 2) of the Lemma and Conditions 1a, 1b, and 1d of Definition 7 are satisfied for the composed state $\widehat{s}$. To prove Condition 1c it suffices to observe that due to Line 3 and the fact that $c \equiv_2 \alpha_s$, it holds that $\mathsf{H}^\beta_{\widehat{s}} = \mathsf{H}^\beta_s$ for any $\beta \in \mathbb{B}$. In particular, we have that $\Delta^{\alpha_s,q}_{\widehat{\mathsf{r}}} = \Delta^{\alpha_s,q}_\mathsf{r}$ for all priorities $q \geq p$, $\Delta^{\overline{\alpha}_s,q}_{\widehat{\mathsf{u}}} = \Delta^{\overline{\alpha}_s,q}_\mathsf{u}$ for all priorities $q > c$ and $\Delta^{\overline{\alpha}_s,c}_{\widehat{\mathsf{u}}} = \Delta^{\overline{\alpha}_s,c}_\mathsf{u} \cup \mathsf{U}_s$. Finally, if $\widehat{s}$ is *closed*, it necessarily holds that $\widehat{s}$ is $\overline{\alpha}_s$-maximal and that $\mathsf{R}_{\widehat{s}}$ is α-maximal as a direct consequence of the fact that s is strongly-maximal and $\mathsf{r}^{(>p)\wedge(<c_s)} = \emptyset$. Hence, $\widehat{s}$ is also promotable. We can now focus on the case in which $c \equiv_2 \overline{\alpha}_s$. By Lines 1 and 3-4 of Algorithm Und we have that $\widehat{\mathsf{u}} = \mathsf{u}^{(\geq c)}[\mathsf{L}_s \mapsto c]$ and $\widehat{\mathsf{r}} = \mathsf{r}^{(\geq c)}[v \in \mathsf{U}_s \mapsto \mathsf{pr}(v)]$. Now, it is easy to see that also in this case both Points 1) and 2) of the Lemma and Conditions 1a, 1b, and 1d are satisfied for the composed state $\widehat{s}$, since $\mathsf{L} \triangleq \mathsf{dom}\big(\mathsf{r}^{\leq p}\big)$ and $\mathsf{U} \triangleq \mathsf{u}^{-1}(p)$ and, it also holds that $\widehat{\mathsf{u}}$ is a promotion function and $\widehat{\mathsf{r}}$ is a region function due to the fact that the priority function pr is always a region function. Condition 1c, instead, follows from the fact that s is a strongly-maximal hybrid state and, therefore, it holds that $\mathsf{esc}^\alpha(\mathsf{L}_s) \subseteq \mathsf{H}^{\overline{\alpha}_s}_s$. By the same Condition 1c, we have that $\mathsf{esc}^{\alpha_s}(\Delta^{\alpha_s,p}_\mathsf{u}) \subseteq \mathsf{pre}^{\overline{\alpha}_s}(\Delta^{\alpha_s,p}_\mathsf{r} \setminus \mathsf{L}_s)$ and, consequently, for each priority $q > p$ the set $\mathsf{u}^{-1}(q)$ cannot reach U_s that, due to Line 5, is reset to the values of the priority function pr. The latter observation also implies that, if $\widehat{s}$ is *closed*, $\widehat{s}$ is also $\overline{\alpha}_s$-maximal. Indeed, when $\widehat{s}$ is *closed*, it holds that player $\overline{\alpha}_s$ cannot attract $\mathsf{R}_{\widehat{s}}$ and since $\mathsf{L}_{\widehat{s}} \supseteq \mathsf{R}_{\widehat{s}}$ we can conclude that $\mathsf{H}^{\alpha_s}_{\widehat{s}} \setminus \mathsf{L}_{\widehat{s}}$ is $\overline{\alpha}_s$-maximal *w.r.t.* $\mathsf{L}_{\widehat{s}}$. Finally, similarly to the previous case, $\mathsf{R}_{\widehat{s}}$ turns out to be α-maximal as a direct consequence of the fact that s is strongly-maximal and that $\mathsf{r}^{(>p)\wedge(<c_s)} = \emptyset$. Hence, $\widehat{s}$ is also promotable. □

Lemma 13. *Given a maximal hybrid state $s \in \mathsf{St}_\mathsf{M}$, the function* **sol**$(s)$ *terminates and returns a pair $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \in \mathsf{Rg} \times \mathsf{Pf}$ of a region function $\widehat{\mathsf{r}}$ and a promotion function $\widehat{\mathsf{u}}$ such that: 1) $\mathsf{r}_s^{(>p_s)} \subseteq \widehat{\mathsf{r}}^{(>p_s)}$; 2) $\mathsf{u}_s^{(>p_s)} \subseteq \widehat{\mathsf{u}}^{(>p_s)}$; 3) $\widehat{s} = ((\widehat{\mathsf{r}}, c_s), (\widehat{\mathsf{u}}, \mathsf{b}_0, \mathsf{b}_1), q)$ is a promotable hybrid state if $\widehat{s}$ is* closed *and, an hybrid state otherwise, for all priorities $q \in \mathsf{pr}$ such that $c_s < q \leq \mathsf{min}(\mathsf{rng}(\mathsf{r}_s \cup \mathsf{u}_s{}^{>c_s}))$, and $\mathsf{b}_0, \mathsf{b}_1 \in \mathbb{N}$. Moreover, the procedure* **hsol**(s) *modifies in-place s into a strongly-maximal hybrid state $\widehat{s} \in \mathsf{St}_\mathsf{S}$ such that $\widehat{s} \preccurlyeq s$.*

Proof. Let $\widehat{s} \triangleq$ **sol**(s) be the result obtained by computing **sol** on a maximal hybrid state $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, b_0, b_1), c)$. The proof proceeds by induction, where in the base case we have that $p = \bot$ or $b_{\alpha_s} = 0$. In this case, by Line 1 and 2 of Algorithm 5 we

have that $\widehat{s} = ((\mathsf{r}, c_s), (\mathsf{u}, \mathtt{b}_0, \mathtt{b}_1), q)$, which trivially satisfies all the requirements. Therefore, let us now consider an arbitrary recursive call of **sol** where $p > \bot$ and $b_{\alpha_s} > 0$. At Line 3 of **sol**, the procedure **hsol** is called on the state s. Consequently, at Line 1 of Algorithm 9, the set R_s is maximised by computing its α_s-attractor $\mathsf{atr}^{\alpha_s}_{\supset_s}$. It is not hard to show that the new state s is still a strongly-maximal hybrid state greater or equal to the old one, *w.r.t.* the ordering of Definition 7. Then, two cases may arise at Line 4 of Half: s is *open* or *closed*. In the first case, at Line 5 a recursive call of **sol** is performed on the state generated by Half. Now, due to the fact that s is a strongly maximal hybrid state and by Lemma 3, it follows that the resulting state of $\mathsf{Half}(s)$ is a maximal hybrid state greater or equal to the input s, *w.r.t.* the ordering of Definition 7. Hence, by external induction, we have that $s = ((\widehat{\mathsf{r}}, p), (\widehat{\mathsf{u}}, b_0, b_1), c)$ is a hybrid state and, by Point 1) and 2) it is also greater or equal to the previous s, *w.r.t.* the ordering of Definition 7. At this point (Line 6 of Half), s can be *open* or *closed*. Let us first consider again the case in which it is *open*. By Lemma 4 and Line 7, the function Maximise modifies s into a maximal hybrid state that is greater or equal to the previous s, *w.r.t.* the ordering of Definition 7. On the other hand, when s is *closed*, by external induction it holds that s is promotable and, by Lemma 5 the function Promote modifies s into a maximal hybrid state greater than the previous one *w.r.t.* the ordering of Definition 7. This last case also applies when s is *closed* at Line 4. Now, since the modified state by $\mathsf{Promote}(s)$ is always greater than s, *w.r.t.* the ordering of Definition 7, we can conclude that function **hsol** only ends when the function Maximise modifies the input hybrid state such that there is no progress *w.r.t.* the ordering of Definition 7, otherwise **hsol** starts a new iteration due to the Loop at Line 1. By the fact that the state space is finite and that it starts a new iteration only if there is a progress in the ordering it is easy to prove that **hsol** always terminates. Observe that, when $s' \not\prec s$ where s' is the state modified by $\mathsf{Maximise}(s)$, it holds that s' is also strongly-maximal and, therefore, **hsol** returns a strongly maximal hybrid state. We can now go back to Line 3 of Algorithm 5 where the modified state s is strongly maximal as observed above. Now, by Lemma 3 the input state of **sol** at Line 5 is maximal. Hence, by external induction, the modified state composed of the results of Line 5 is $s = ((\widehat{\mathsf{r}}, p), (\widehat{\mathsf{u}}, b_0, b_1), c)$ that is a hybrid state and, by Point 1) and 2) it is also greater or equal to the previous state stored as $\widehat{s}$, *w.r.t.* the ordering of Definition 7. Similarly to the proof for **hsol**, the state s can be *open* or *closed*. In the first case, two more cases may arise, indeed, if s is *open*, then by Lemma 4 and Line 7, the function Maximise modifies s into a maximal hybrid state that is greater or equal to the previous s, *w.r.t.* the ordering of Definition 7. On the other hand, if s is *closed*, then by external induction it holds that s is promotable and, by Lemma 5 the function Promote modifies s into a maximal hybrid state greater than the previous one *w.r.t.* the ordering of Definition 7. Thus, at Line 9, in case $s \prec \widehat{s}$, where $\widehat{s}$ is the state stored before the recursive calls at Line 4, **hsol** is called on a maximal state and, as proved above, the modified resulting state is strongly maximal. Otherwise, as observed in the proof for **hsol**, s is already strongly-maximal. As a consequence, by Lemma 6 and Line 10, we can conclude that **sol** returns a pair of region function and priority function $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}})$ such that $\widehat{s} = ((\widehat{\mathsf{r}}, c_s), (\widehat{\mathsf{u}}, \mathtt{b}_0, \mathtt{b}_1), q)$ is a promotable hybrid state if $\widehat{s}$ is *closed* and, an hybrid state otherwise, for all priorities $q \in \mathsf{pr}$ such that $c_s < q \leq \mathsf{min}(\mathsf{rng}(\mathsf{r}_s \cup \mathsf{u}_s{}^{>c_s}))$, and $\mathtt{b}_0, \mathtt{b}_1 \in \mathbb{N}$. □

References

[1] E. Emerson, C. Jutla, A. Sistla, On model checking for the muCalculus and its fragments, Theor. Comput. Sci. 258 (1–2) (2001) 491–522.
[2] E. Emerson, C. Lei, Efficient model checking in fragments of the propositional muCalculus, in: Logic in Computer Science'86, IEEE Computer Society, 1986, pp. 267–278.
[3] M. Vardi, Reasoning about the past with two-way automata, in: International Colloquium on Automata, Languages, and Programming'98, in: LNCS, vol. 1443, Springer, 1998, pp. 628–641.
[4] N. Piterman, From nondeterministic Büchi and streett automata to deterministic parity automata, in: Logic in Computer Science'06, IEEE Computer Society, 2006, pp. 255–264.
[5] S. Schewe, Tighter bounds for the determinisation of Büchi automata, in: Foundations of Software Science and Computational Structures'09, in: LNCS, vol. 5504, Springer, 2009, pp. 167–181.
[6] E. Grädel, W. Thomas, T. Wilke, Automata, Logics, and Infinite Games: A Guide to Current Research, LNCS, vol. 2500, Springer, 2002.
[7] O. Kupferman, M. Vardi, Weak alternating automata and tree automata emptiness, in: Symposium on Theory of Computing'98, Association for Computing Machinery, 1998, pp. 224–233.
[8] K. Chatterjee, T. Henzinger, N. Piterman, Strategy logic, Inf. Comput. 208 (6) (2010) 677–693.
[9] F. Mogavero, A. Murano, M. Vardi, Reasoning about strategies, in: Foundations of Software Technology and Theoretical Computer Science'10, in: LIPIcs, vol. 8, Leibniz-Zentrum fuer Informatik, 2010, pp. 133–144.
[10] F. Mogavero, A. Murano, G. Perelli, M. Vardi, What makes ATL* decidable? A decidable fragment of strategy logic, in: Concurrency Theory'12, in: LNCS, vol. 7454, Springer, 2012, pp. 193–208.
[11] M. Benerecetti, F. Mogavero, A. Murano, Substructure temporal logic, in: Logic in Computer Science'13, IEEE Computer Society, 2013, pp. 368–377.
[12] D. Berwanger, E. Grädel, Fixed-point logics and solitaire games, Theor. Comput. Sci. 37 (6) (2004) 675–694.
[13] D. Kozen, Results on the propositional muCalculus, Theor. Comput. Sci. 27 (3) (1983) 333–354.
[14] L. de Alfaro, T. Henzinger, R. Majumdar, From verification to control: dynamic programs for omega-regular objectives, in: Logic in Computer Science'01, IEEE Computer Society, 2001, pp. 279–290.
[15] T. Wilke, Alternating tree automata, parity games, and modal muCalculus, Bull. Belg. Math. Soc. 8 (2) (2001) 359–391.
[16] S. Schewe, B. Finkbeiner, Satisfiability and finite model property for the alternating-time muCalculus, in: Computer Science Logic'06, in: LNCS, vol. 4207, Springer, 2006, pp. 591–605.
[17] R. Alur, T. Henzinger, O. Kupferman, Alternating-time temporal logic, J. ACM 49 (5) (2002) 672–713.
[18] S. Schewe, ATL* satisfiability is 2ExpTime-complete, in: International Colloquium on Automata, Languages, and Programming'08, in: LNCS, vol. 5126, Springer, 2008, pp. 373–385.
[19] A. Mostowski, Games with Forbidden Positions, Tech. Rep., University of Gdańsk, Gdańsk, Poland, 1991.
[20] E. Emerson, C. Jutla, Tree automata, muCalculus, and determinacy, in: Foundation of Computer Science'91, IEEE Computer Society, 1991, pp. 368–377.
[21] A. Martin, Borel Determinacy, Ann. Math. 102 (2) (1975) 363–371.
[22] M. Jurdziński, Deciding the winner in parity games is in UP ∩ co-UP, Inf. Process. Lett. 68 (3) (1998) 119–124.

[23] C. Calude, S. Jain, B. Khoussainov, W. Li, F. Stephan, Deciding parity games in quasipolynomial time, in: Symposium on Theory of Computing'17, Association for Computing Machinery, 2017, pp. 252–263.
[24] M. Benerecetti, D. Dell'Erba, F. Mogavero, Solving parity games via priority promotion, in: Computer Aided Verification'16, in: LNCS, vol. 9780 (Part II), Springer, 2016, pp. 270–290.
[25] M. Benerecetti, D. Dell'Erba, F. Mogavero, A delayed promotion policy for parity games, in: Games, Automata, Logics, and Formal Verification'16, in: EPTCS, vol. 226, 2016, pp. 30–45.
[26] M. Benerecetti, D. Dell'Erba, F. Mogavero, Improving priority promotion for parity games, in: Haifa Verification Conference'16, in: LNCS, vol. 10028, Springer, 2016, pp. 1–17.
[27] M. Benerecetti, D. Dell'Erba, F. Mogavero, Solving parity games via priority promotion, Form. Methods Syst. Des. 52 (2) (2018) 193–226.
[28] M. Benerecetti, D. Dell'Erba, F. Mogavero, A delayed promotion policy for parity games, Inf. Comput. 262 (2) (2018) 221–240.
[29] R. McNaughton, Infinite games played on finite graphs, Ann. Pure Appl. Log. 65 (1993) 149–184.
[30] W. Zielonka, Infinite games on finitely coloured graphs with applications to automata on infinite trees, Theor. Comput. Sci. 200 (1–2) (1998) 135–183.
[31] W. Ludwig, A subexponential randomized algorithm for the simple stochastic game problem, Inf. Comput. 117 (1) (1995) 151–155.
[32] A. Puri, Theory of Hybrid Systems and Discrete Event Systems, Ph.D. thesis, University of California, Berkeley, CA, USA, 1995.
[33] J. Vöge, M. Jurdziński, A discrete strategy improvement algorithm for solving parity games, in: Computer Aided Verification'00, in: LNCS, vol. 1855, Springer, 2000, pp. 202–215.
[34] H. Björklund, S. Vorobyov, A combinatorial strongly subexponential strategy improvement algorithm for mean-payoff games, Discrete Appl. Math. 155 (2) (2007) 210–229.
[35] S. Schewe, An optimal strategy improvement algorithm for solving parity and payoff games, in: Computer Science Logic'08, in: LNCS, vol. 5213, Springer, 2008, pp. 369–384.
[36] J. Fearnley, Non-oblivious strategy improvement, in: Logic for Programming Artificial Intelligence and Reasoning'10, in: LNCS, vol. 6355, Springer, 2010, pp. 212–230.
[37] S. Schewe, A. Trivedi, T. Varghese, Symmetric strategy improvement, in: International Colloquium on Automata, Languages, and Programming'15, in: LNCS, vol. 9135, Springer, 2015, pp. 388–400.
[38] O. Friedmann, An exponential lower bound for the latest deterministic strategy iteration algorithms, Log. Methods Comput. Sci. 7 (3) (2011) 1–42.
[39] O. Friedmann, A subexponential lower bound for Zadeh's pivoting rule for solving linear programs and games, in: Integer Programming and Combinatorial Optimization'11, in: LNCS, vol. 6655, Springer, 2011, pp. 192–206.
[40] O. Friedmann, A superpolynomial lower bound for strategy iteration based on snare memorization, Discrete Appl. Math. 161 (10–11) (2013) 1317–1337.
[41] T. van Dijk, G. Loho, M. Maat, The worst-case complexity of symmetric strategy improvement, in: Computer Science Logic'24, in: LIPIcs, vol. 288, Leibniz-Zentrum fuer Informatik, 2024, pp. 24:1–24:19.
[42] O. Friedmann, Recursive algorithm for parity games requires exponential time, RAIRO Theor. Inform. Appl. 45 (4) (2011) 449–457.
[43] M. Benerecetti, D. Dell'Erba, F. Mogavero, Robust exponential worst cases for divide-et-impera algorithms for parity games, in: Games, Automata, Logics, and Formal Verification'17, in: EPTCS, vol. 256, 2017, pp. 121–135.
[44] M. Benerecetti, D. Dell'Erba, F. Mogavero, Robust worst cases for parity games algorithms, Inf. Comput. 272 (104501) (2020) 1–31.
[45] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theor. Comput. Sci. 158 (1–2) (1996) 343–359.
[46] A. Browne, E. Clarke, S. Jha, D. Long, W. Marrero, An improved algorithm for the evaluation of fixpoint expressions, Theor. Comput. Sci. 178 (1–2) (1997) 237–255.
[47] M. Jurdziński, Small progress measures for solving parity games, in: Symposium on Theoretical Aspects of Computer Science'00, in: LNCS, vol. 1770, Springer, 2000, pp. 290–301.
[48] S. Schewe, Solving parity games in big steps, J. Comput. Syst. Sci. 84 (2017) 243–262.
[49] M. Jurdziński, M. Paterson, U. Zwick, A deterministic subexponential algorithm for solving parity games, SIAM J. Comput. 38 (4) (2008) 1519–1532.
[50] J. Fearnley, S. Jain, S. Schewe, F. Stephan, D. Wojtczak, An ordered approach to solving parity games in quasi polynomial time and quasi linear space, in: SPIN Symposium on Model Checking of Software'17, Association for Computing Machinery, 2017, pp. 112–121.
[51] M. Jurdziński, R. Lazic, Succinct progress measures for solving parity games, in: Logic in Computer Science'17, Association for Computing Machinery, 2017, pp. 1–9.
[52] D. Dell'Erba, S. Schewe, Smaller progress measures and separating automata for parity games, Front. Comput. Sci. 4 (8) (2022) 1–18.
[53] K. Lehtinen, A modal mu perspective on solving parity games in quasi-polynomial time, in: Logic in Computer Science'18, Association for Computing Machinery & IEEE Computer Society, 2018, pp. 639–648.
[54] P. Parys, Parity games: another view on Lehtinen's algorithm, in: Computer Science Logic'20, in: LIPIcs, vol. 152, Leibniz-Zentrum fuer Informatik, 2020, pp. 32:1–32:15.
[55] L. Daviaud, M. Jurdzinski, K. Thejaswini, The Strahler number of a parity game, in: International Colloquium on Automata, Languages, and Programming'20, in: LIPIcs, vol. 168, Leibniz-Zentrum fuer Informatik, 2020, pp. 123:1–123:19.
[56] P. Parys, Parity games: Zielonka's algorithm in quasi-polynomial time, in: Mathematical Foundations of Computer Science'19, in: LIPIcs, vol. 138, Leibniz-Zentrum fuer Informatik, 2019, pp. 10:1–10:13.
[57] W. Czerwinski, L. Daviaud, N. Fijalkow, M. Jurdzinski, R. Lazic, P. Parys, Universal trees grow inside separating automata: quasi-polynomial lower bounds for parity games, in: Symposium on Discrete Algorithms'19, SIAM, 2019, pp. 2333–2349.
[58] K. Lehtinen, S. Schewe, D. Wojtczak, Improving the complexity of Parys' recursive algorithm, Tech. Rep., arXiv:1904.11810, 2019.
[59] K. Lehtinen, P. Parys, S. Schewe, D. Wojtczak, A recursive approach to solving parity games in quasipolynomial time, Log. Methods Comput. Sci. 18 (1) (2022).
[60] M. Benerecetti, D. Dell'Erba, M. Faella, F. Mogavero, From quasi-dominions to progress measures, in: Automata Theory and Applications: Games, Learning and Structures, World Scientific, 2023, pp. 159–199.
[61] M. Benerecetti, D. Dell'Erba, F. Mogavero, Solving mean-payoff games via quasi dominions, in: Tools and Algorithms for the Construction and Analysis of Systems'20, in: LNCS, vol. 12079, Springer, 2020, pp. 289–306.
[62] M. Benerecetti, D. Dell'Erba, F. Mogavero, Solving mean-payoff games via quasi dominions, Inf. Comput. 297 (105151) (2024) 1–25.
[63] M. Jurdziński, M. Paterson, U. Zwick, A deterministic subexponential algorithm for solving parity games, in: Symposium on Discrete Algorithms'06, SIAM, 2006, pp. 117–123.
[64] T. van Dijk, Oink: an implementation and evaluation of modern parity game solvers, in: Tools and Algorithms for the Construction and Analysis of Systems'18, in: LNCS, vol. 10805, Springer, 2018, pp. 291–308.
[65] Y. Liu, Z. Duan, C. Tian, An improved recursive algorithm for parity games, in: Theoretical Aspects of Software Engineering'14, IEEE Computer Society, 2014, pp. 154–161.
[66] T. van Dijk, Attracting tangles to solve parity games, in: Computer Aided Verification'18, in: LNCS, vol. 10982, Springer, 2018, pp. 198–215.
[67] R. Lapauw, M. Bruynooghe, M. Denecker, Improving parity game solvers with justifications, in: Verification, Model Checking, and Abstract Interpretation'20, in: LNCS, vol. 11990, Springer, 2020, pp. 449–470.
[68] L. Sanchez, W. Wesselink, T.A.C. Willemse, A comparison of BDD-based parity game solvers, in: Games, Automata, Logics, and Formal Verification'18, in: EPTCS, vol. 277, 2018, pp. 103–117.

[69] T. van Dijk, A parity game tale of two counters, in: Games, Automata, Logics, and Formal Verification'19, in: EPTCS, vol. 305, 2019, pp. 107–122.
[70] J. Keiren, Benchmarks for parity games, in: Fundamentals of Software Engineering'15, in: LNCS, vol. 9392, Springer, 2015, pp. 127–142.
[71] C. Koymans, J. Mulder, A Modular Approach to Protocol Verification Using Process Algebra, in: Applications of Process Algebra, Cambridge University Press, 1990, pp. 261–306.
[72] V. Cerf, R. Kahn, A protocol for packet network intercommunication, Trans. Commun. 22 (5) (1974) 637–648.
[73] J. Groote, J. van de Pol, A Bounded Retransmission Protocol for Large Data Packets, American Mathematical Society Translations96, LNCS, vol. 1101, Springer, 1996, pp. 536–550.
[74] K. Bartlett, R. Scantlebury, P. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, Commun. ACM 12 (5) (1969) 260–261.
[75] R. Veldema, R. Hofman, R. Bhoedjang, C. Jacobs, H. Bal, Source-level global optimizations for fine-grain distributed shared memory systems, in: Principles and Practice of Parallel Programming'01, Association for Computing Machinery, 2001, pp. 83–92.
[76] W. Hesselink, Invariants for the construction of a handshake register, Inf. Process. Lett. 68 (4) (1998) 173–177.
[77] O. Friedmann, M. Lange, Solving parity games in practice, in: Automated Technology for Verification and Analysis'09, in: LNCS, vol. 5799, Springer, 2009, pp. 182–196.
[78] O. Friedmann, M. Lange, A solver for modal fixpoint logics, Electron. Notes Theor. Comput. Sci. 262 (2010) 99–111.