Priority Promotion with Parysian Flair

Massimo Benerecetti[1], Daniele Dell'Erba[2], Fabio Mogavero[1], Sven Schewe[2], and Dominik Wojtczak[2]

[1] Università degli Studi di Napoli Federico II, Naples, Italy
{massimo.benerecetti, fabio.mogavero}@unina.it
[2] University of Liverpool, Liverpool, UK
{daniele.dell-erba,sven.schewe,d.wojtczak}@liverpool.ac.uk

Abstract. We develop an algorithm that combines the advantages of priority promotion – the leading approach 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.

1 Introduction

Parity games have many applications in model checking [38,18,16,1,64,39] and synthesis [64,38,61,55,50,57,58]. In particular, modal and alternating-time μ-calculus model checking [64,1], synthesis [58,50,57] and satisfiability checking [64,38,61,55] for reactive systems, module checking [39], and ATL* model checking [16,1] can be reduced to solving parity games. This relevance of parity games led to a series of different approaches to solving them [45,19,44,51,66,12,65,32,33,63,46,40,9,10,35,22,7,13,34,41,23,48,42,49,15].

The research falls into two categories: to develop *fast solvers*; to determine the *complexity of parity games* or to find algorithms with a good *worst-case complexity*. With its practical motivation, the leading algorithms most for solving real life parity games are currently *priority promotion* techniques [7,6], a refinement of the classic *recursive algorithm* [45,19,65] that follows the iterated fixed-point structure induced by the parity condition. The complexity of solving parity games is still an open problem. Parity games are *memoryless determined* [18,11], which implies that nondeterministic algorithms can determine winning regions and strategies for both players. Due to their symmetry, they are therefore in NPTime ∩ CoNPTime [18], and by reduction to payoff games [66], in UPTime ∩ CoUPTime [32]. While determining their membership in PTime continues to be a major challenge, one of the most celebrated results in recent years has been the landmark result of Calude *et al.* [13], 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 [45,19,66,12,65,33,10,53,54,7] ($n^{O(c)}$), or in the square-root of the number of game positions [44,35,10] (approximately $n^{O(\sqrt{n})}$). The breakthrough of Calude *et al.* [13] has triggered a new line of research into QP algorithms, including [34,41,23,48,42,49,15].

Algorithms that are good in practice do not tend to display their worst-case behaviour, except for in carefully designed hostile examples. This holds in particular for strategy improvement algorithms [44,51,63,10,53,20,56], which were considered candidates for tractable algorithms until they were shown to be exponential by Friedman's delicate lower bound constructions [24,29,26] (with the notable exception of the symmetric approach from [56], for which no hard families are known). But while it is easier to design hard classes for recursive [25,5] and priority-promotion algorithms [6], these classes are still not relevant in practice. However, with a host of QP algorithms at hand, an upgrade of priority promotion that offers QP lower bounds without undue compromise on efficiency will be an attractive challenge that combines the best of both worlds.

Interestingly, Parys' algorithm [48] and variations thereof [42], which, like priority promotion techniques, adjust the classic recursive algorithm [45,19,65], are relatively fast among the QP algorithms, where [48] has the edge on benchmarks, while [42] has the edge on guarantees. On first glance, this seems to invite synthesising one of these algorithms with priority promotion. On second glance, the prospect of this synthesis seems less promising. Priority promotion techniques [3,7,6] achieve their advancement over the previously leading recursive algorithms [45,19,65] 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 with quickly falling strength 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 structure of [7] and [48]: 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 $\mathcal{A} = \langle \mathrm{Ps}_{0}, \mathrm{Ps}_{1}, Mv \rangle$, with $\mathrm{Ps}_{0} \cap \mathrm{Ps}_{1} = \emptyset$ and $\mathrm{Ps} \triangleq \mathrm{Ps}_{0} \cup \mathrm{Ps}_{1}$, such that $\langle \mathrm{Ps}, Mv \rangle$ is a finite directed graph without sinks. Ps_{0} (*resp.*, Ps_{1}) is the set of positions of the Player (*resp.*, the Opponent) and $Mv \subseteq \mathrm{Ps} \times \mathrm{Ps}$ is a left-total relation describing all possible moves. A *path* in $\mathrm{V} \subseteq \mathrm{Ps}$ is a finite or infinite sequence $\pi \in \mathrm{Pth}(\mathrm{V})$ of positions in V compatible with the move relation, *i.e.*, $(\pi_i, \pi_{i+1}) \in Mv$, for all $i \in [0, |\pi| - 1)$. 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.*, $(v, \sigma_\alpha(v)) \in Mv$. With $\mathrm{Str}_\alpha(\mathrm{V})$ we denote the set of all α-strategies on V. A *play* in $\mathrm{V} \subseteq \mathrm{Ps}$ from a position $v \in \mathrm{V}$ *w.r.t.* a pair of strategies $(\sigma_0, \sigma_1) \in \mathrm{Str}_0(\mathrm{V}) \times \mathrm{Str}_1(\mathrm{V})$, called $((\sigma_0, \sigma_1), v)$-*play*, is a path $\pi \in \mathrm{Pth}(\mathrm{V})$ such that $(\pi)_0 = v$ and, for all $i \in [0, |\pi| - 1)$, if $(\pi)_i \in \mathrm{Ps}_0$

then $(\pi)_{i+1} = \sigma_0((\pi)_i)$ else $(\pi)_{i+1} = \sigma_1((\pi)_i)$. The *play function* $\mathsf{play} : (\mathrm{Str}_0(\mathrm{V}) \times \mathrm{Str}_1(\mathrm{V})) \times \mathrm{V} \to \mathrm{Pth}(\mathrm{V})$ returns, for each position $v \in \mathrm{V}$ and pair of strategies $(\sigma_0, \sigma_1) \in \mathrm{Str}_0(\mathrm{V}) \times \mathrm{Str}_1(\mathrm{V})$, the maximal $((\sigma_0, \sigma_1), v)$-play $\mathsf{play}((\sigma_0, \sigma_1), v)$.

A *parity game* is a tuple $\partial = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathsf{PG}$, 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: $\mathsf{pr}(\partial) \triangleq \mathsf{max}_{v \in \mathrm{Ps}} \mathsf{pr}(v)$; for a path $\pi \in \mathrm{Pth}$, we set $\mathsf{pr}(\pi) \triangleq \mathsf{max}_{i \in [0, |\pi|)} \mathsf{pr}((\pi)_i)$, if π is finite, and $\mathsf{pr}(\pi) \triangleq \limsup_{i \in \mathbb{N}} \mathsf{pr}((\pi)_i)$, otherwise. A set of positions $\mathrm{V} \subseteq \mathrm{Ps}$ is an α-*dominion*, with $\alpha \in \{0, 1\}$, if there exists an α-strategy $\sigma_\alpha \in \mathrm{Str}_\alpha(\mathrm{V})$ such that, for all $\overline{\alpha}$-strategies $\sigma_{\overline{\alpha}} \in \mathrm{Str}_{\overline{\alpha}}(\mathrm{V})$ and positions $v \in \mathrm{V}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ is infinite and $\mathsf{pr}(\pi) \equiv_2 \alpha$. In other words, σ_α only induces on V infinite plays whose maximal priority visited infinitely often has parity α. The *winning region* for player $\alpha \in \{0, 1\}$ in game ∂, denoted by $\mathrm{Wn}^\alpha_\partial$, is the greatest set of positions that is also a α-dominion in ∂. Since parity games are memoryless determined [17], meaning that from each position one of the two players wins, the two winning regions of a game ∂ form a partition of its positions, *i.e.*, $\mathrm{Wn}^0_\partial \cup \mathrm{Wn}^1_\partial = \mathrm{Ps}_\partial$.

By $\partial \setminus \mathrm{V}$ we denote the maximal subgame of ∂ 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 $\mathsf{pre}^\alpha(\mathrm{V}) \triangleq \{ v \in \mathrm{Ps}_\alpha \mid Mv(v) \cap \mathrm{V} \neq \emptyset \} \cup \{ v \in \mathrm{Ps}_{\overline{\alpha}} \mid Mv(v) \subseteq \mathrm{V} \}$, 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 $\mathsf{pre}^\alpha(\mathrm{V})$ to an arbitrary number of moves. Thus, it corresponds to the least fix-point of that operator. When $\mathrm{V} = \mathsf{pre}^\alpha(\mathrm{V})$, 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 $\partial \setminus \mathrm{V}$ 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 ∂ they are applied to. In the rest of the paper, we shall only add ∂ as subscript of an operator, *e.g.* $\mathsf{atr}^\alpha_\partial(\mathrm{V})$, when the game is not clear from the context.

3 A Hybrid Priority-Promotion Algorithm

We introduce the hybrid algorithm in three steps. In the first step (Section 3.1), we introduce a variation of classic Priority Promotion, which serves as the backbone of our hybrid algorithm in Section 3.3. We provide a recap of how Priority Promotion operates and an introduction to the data structure that is later extended. In a nutshell, Priority Promotion accelerates the classic recursive algorithm, by allowing to merge dominions in subgames spanning non-adjacent recursive calls, which is the essence of the promotion operations. In the following subsection (Section 3.2), we outline Parys' algorithm, which does not seek to identify all dominions on a level, but merely those 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 or 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 are set to fail, when the bounds are larger (and thus the required guarantees stronger) along the call tree. In Section 3.3, 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' (which works across levels because dominions have unbounded guarantees). The identification of the right set to promote to, instead, remains fairly similar to the way it 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, 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.

3.1 The Priority-Promotion Approach

The *priority-promotion approaches* [7,6] attack the problem of solving a parity game $\eth \in \mathsf{PG}$ by iteratively computing, one at a time, a sequence of α-dominions $\mathrm{D}^{\alpha}_{0}, \mathrm{D}^{\alpha}_{1}, \ldots \subseteq \mathrm{Ps}$, for some player $\alpha \in \mathbb{B} \triangleq \{0, 1\}$. These, indeed, are portions of the two winning regions, Wn_0 and Wn_1, that need to be identified. The idea here is to start from a weaker notion, called *quasi dominion*, and then compose them until a dominion is obtained. The name of the approach comes precisely from the fact that this composition is computed by applying the following operation of *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.

Similarly to a dominion, a quasi dominion is a set of positions over which one of the two players, called the leading player, has a strategy defined on that set, whose induced plays, if infinite, are winning for that player. As opposed to dominions, however, some of these plays may be finite, since the opponent may have the possibility to escape from those positions towards a different part of the game, hoping for a better outcome.

Definition 1 (Quasi Dominion). *A set of positions* $\mathrm{Q} \subseteq \mathrm{Ps}$ *is a* quasi α-dominion*, for some player* $\alpha \in \{0, 1\}$*, if there exists an* α*-strategy* $\sigma_\alpha \in \mathrm{Str}^{\alpha}(\mathrm{Q})$*, called* α-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* $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$*, namely a* (σ_α, v)-play*, 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 closed subset of a quasi dominion is found, a dominion is identified.

Theorem 1 (Induced Dominion). *Let $\alpha \in \{0, 1\}$ 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 $Mv(v) \subseteq \mathrm{D}$, otherwise, for all positions $v \in \mathrm{D}$. Then, D is an α-dominion.*

Intuitively, the solution algorithms following this approach carry on the search for a dominion by exploring a *finite strict partial order* $\langle \mathrm{St}, s_I, \prec \rangle$, whose elements, called *states*, record information about the quasi dominions computed up to a certain 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.*, it if is not an α-dominion. If, on the other hand, it is *closed*, the search is over and Q is added to the portion of the winning region Wn_α computed so far.

We start by describing a new 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 first need to introduce few technical notions, all of which refer to some fixed parity game $\supseteq \in \mathsf{PG}$. 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 $\supseteq$ extended with the bottom symbol $\bot$ and two top symbols $\top_0$ and $\top_1$, one for each player. 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 *promotion function*, which represents the backbone of the algorithm, being the data structure to which the promotion operation is applied. Intuitively, it is a partial function from positions to priorities that over-approximates the priority function of the game.

Definition 2 (Promotion Function). *A* promotion function $\mathsf{r} \in \mathrm{Pm} \triangleq \mathrm{Ps} \rightharpoonup \mathrm{Pr}_\top$ *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 [7]. Given a promotion function $\mathsf{r} \in \mathrm{Pm}$ and a priority $p \in \mathrm{Pr}$, we denote with $\mathsf{r}^{(\sim p)}$, for $\sim \in \{<, \leq, \equiv_2, \geq, >\}$, the function obtained by restricting the domain of r to those positions $v \in \mathsf{dom}(\mathsf{r})$ whose priority $\mathsf{r}(v)$ satisfies the relation $\mathsf{r}(v) \sim p$, *i.e.*, $\mathsf{r}^{(\sim p)} \triangleq \mathsf{r} \upharpoonright \{ v \in \mathsf{dom}(\mathsf{r}) \mid \mathsf{r}(v) \sim p \}$, where $\upharpoonright$ is the standard operation of domain restriction. We may also use Boolean combinations of the above restrictions, as in $\mathsf{r}^{(\equiv_2 \alpha) \wedge (\geq p)}$. By $\mathrm{H}_{\mathsf{r}}^{\alpha} \triangleq \mathsf{dom}\big(\mathsf{r}^{(\equiv_2 \alpha)}\big)$ we denote the set of positions in r with a priority congruent to $\alpha \in \mathbb{B}$ and with $\mathrm{H}_{\mathsf{r}}^{\alpha, p} \triangleq \mathsf{dom}\big(\mathsf{r}^{(\equiv_2 \alpha) \wedge (\geq p)}\big)$ its subset with priorities greater than or equal to p.

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

Definition 3 (Quasi-Dominion Function). *A* quasi-dominion function $\mathsf{r} \in \mathrm{Qs} \subseteq \mathrm{Pm}$ *is a promotion function satisfying the following conditions, for every $\alpha \in \mathbb{B}$: 1) the set $\mathrm{H}^{\alpha}_{\mathsf{r}}$ is a quasi α-dominion; 2) the set $\mathsf{r}^{-1}(\top_{\alpha})$ is an α-dominion.*

An important property of dominions is that the extension of an α-dominion by means of its α-attractor 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 subclass of quasi dominions that meets the following requirements: 1) the set $\mathsf{esc}(\mathrm{Q}, \sigma) \triangleq \{ v \in \mathrm{Ps}_{\alpha} \cap \mathrm{Q} \mid \sigma(v) \notin \mathrm{Q} \}$ of α-positions, which leave a quasi α-dominion $\mathrm{Q} \subseteq \mathrm{Ps}$ by following one of its α-witnesses $\sigma \in \mathrm{Str}^{\alpha}(\mathrm{Q})$, is a subset of the $\overline{\alpha}$-escape positions $\mathsf{esc}^{\overline{\alpha}}(\mathrm{Q})$; 2) all these $\overline{\alpha}$-escape positions have priorities congruent to α and greater than the ones of the positions that can be attracted. The first requirement ensures that, to verify whether Q is an α-dominion, it suffices to check for the emptiness of $\mathsf{esc}^{\overline{\alpha}}(\mathrm{Q})$. The second one, instead, can be exploited to regain closure under extension by α-attractor.

Definition 4 (Region Function). *A* region function $\mathsf{r} \in \mathrm{Rg} \subseteq \mathrm{Qs}$ *is a quasi-dominion function satisfying the following conditions, for every $\alpha \in \mathbb{B}$: 1) there exists an α-witness $\sigma_{\alpha} \in \mathrm{Str}^{\alpha}(\mathrm{H}^{\alpha}_{\mathsf{r}})$ for $\mathrm{H}^{\alpha}_{\mathsf{r}}$ such that $\mathsf{esc}(\mathrm{H}^{\alpha,p}_{\mathsf{r}}, \sigma_{\alpha}) \subseteq \mathsf{esc}^{\overline{\alpha}}(\mathrm{H}^{\alpha}_{\mathsf{r}})$, for all $p \in \mathsf{rng}(\mathsf{r})$, with $p \equiv_2 \alpha$; 2) $p \leq \mathsf{pr}(v) \equiv_2 \alpha$, for all $p \in \mathsf{rng}(\mathsf{r})$, with $p \equiv_2 \alpha$, and $v \in \mathsf{esc}^{\overline{\alpha}}(\mathrm{H}^{\alpha,p}_{\mathsf{r}})$.*

Notice that, every set $\mathrm{H}^{\alpha,p}_{\mathsf{r}}$, with $p \in \mathsf{rng}(\mathsf{r})$, is a quasi α-dominion, being a subset of the quasi α-dominion $\mathrm{H}^{\alpha}_{\mathsf{r}}$. Also, it is immediate to see that the priority function pr of a given parity game $\eth$ is always a region function. Indeed, it is trivially a promotion function. Moreover, the positions with a priority of parity α, *i.e.*, $\mathrm{H}^{\alpha}_{\mathsf{pr}}$, form a quasi α-dominion with α-witness any strategy that always chooses to remain inside the set, if allowed by the move relation. Thus, it is a quasi-dominion function as well. Finally, since $\mathrm{H}^{\alpha,p}_{\mathsf{pr}}$ 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 described in the intuitive explanation of the approach. In particular, to account for the current status of the search of a dominion in a game $\eth$, we define a *state* s as a pair, comprising a region function r and a priority p, with the idea that 1) all quasi α-dominions computed so far are contained in $\mathrm{H}^{\alpha,q}_{\mathsf{r}}$, for some $q > p$, 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 has still to be processed. The initial state is composed of the priority function pr of the game and its maximal priority $\mathsf{pr}(\eth)$. Finally, we assume that a state s_1 is lower 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. $\mathrm{S} \subseteq \mathrm{Rg} \times \mathrm{Pr}_{\perp}$ *is the set of all pairs $s \triangleq (\mathsf{r}, p)$, called* states, *where $\mathsf{dom}(\mathsf{r}) = \mathrm{Ps}$; for every state $s \in \mathrm{St}$, we set (i) $\alpha_s \triangleq p \bmod 2$, (ii) $\mathrm{H}^{\alpha}_{s} \triangleq \mathrm{H}^{\alpha}_{\mathsf{r}}$, (iii) $\mathrm{H}^{\alpha,q}_{s} \triangleq \mathrm{H}^{\alpha,q}_{\mathsf{r}}$, (iv) $\mathrm{R}_s \triangleq \mathsf{r}^{-1}(p)$, and (v) $\mathrm{L}_s \triangleq \mathsf{dom}\big(\mathsf{r}^{(\leq p)}\big)$;*

2. $s_I \triangleq (\mathsf{pr}, \mathsf{pr}(\Game))$ *is the* initial state*;*
3. $s_1 \prec s_2$ *if either* $\mathrm{L}_{s_1} \subset \mathrm{L}_{s_2}$ *or* $\mathrm{L}_{s_1} = \mathrm{L}_{s_2}$ *and* $p_{s_1} < p_{s_2}$.

Given a state $s \in \mathrm{St}$, we refer to L_s as the *local area*, *i.e.*, the set of unprocessed positions yet to be analysed. This also includes the quasi α_s-dominion R_s, called *region*, on which the next step of the search will focus. The two quasi dominions H^0_s and H^1_s partition the entire set of positions in the game, while $\mathrm{H}^{0,q}_s$ and $\mathrm{H}^{1,q}_s$ represent the portions of these quasi dominions to which the region function has assigned a priority at least equal to $q \in \mathrm{Pr}$. Notice that the pseudo-priority $\perp$ is used to indicate the situation where all positions have been processed, which corresponds to an empty local area.

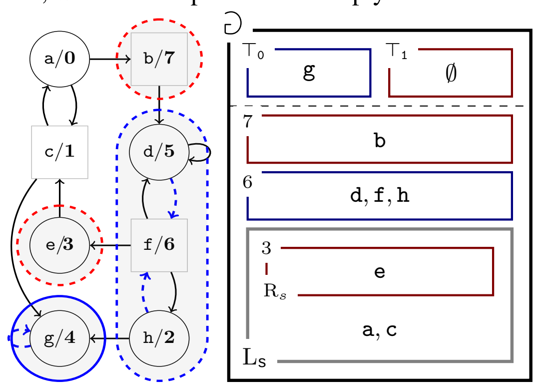

Fig. 1. A game and a corresponding state representation.

To exemplify the above notions, consider the game depicted in Figure 1, where circled 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 the opponent. At the state $s = (\mathsf{r}, 3)$, where $\mathsf{r} = \{\mathtt{a} \mapsto 0; \mathtt{c} \mapsto 1; \mathtt{e} \mapsto 3; \mathtt{d}, \mathtt{f}, \mathtt{h} \mapsto 6; \mathtt{b} \mapsto 7; \mathtt{g} \mapsto \top_0\}$, the local area L_s contains the positions a, c, and e. Of these only e is part of the current region R_s. The quasi 0-dominion H^0_s contains the positions a, d, f, h, and g, while the quasi 1-dominion H^1_s takes the remaining ones, namely b, c, and e. Position g forms a 0-dominion on its own, represented in the picture by the solid closed line. Apart from this position, all the other ones are contained in open quasi dominions, indicated, instead, by the dashed closed lines. For example, the set $\mathsf{r}^{-1}(6) = \{\mathtt{d}, \mathtt{f}, \mathtt{h}\}$ is a quasi 0-dominion, since, if Player 1 decides to remain inside, the adversary wins the play. However, Player 1 also has the choice to escape from position f moving to e, *i.e.*, $\mathsf{esc}^1(\mathsf{r}^{-1}(6)) = \{\mathtt{f}\}$. Similarly, $\mathsf{esc}^0(\mathsf{r}^{-1}(7)) = \{\mathtt{b}\}$. Finally, notice that $\mathrm{H}^{0,4}_s = \{\mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and $\mathrm{H}^{1,4}_s = \{\mathtt{b}\}$.

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. Given a player $\alpha \in \mathbb{B}$, we say that a state $s \in \mathrm{St}$ is α-*maximal*, if the quasi α-dominion $\mathrm{H}^\alpha_s \setminus \mathrm{L}_s$ is α-maximal *w.r.t.* L_s, *i.e.*, the α-attractor $\mathsf{atr}^\alpha(\mathrm{H}^\alpha_s \setminus \mathrm{L}_s, \mathrm{L}_s)$ to $\mathrm{H}^\alpha_s \setminus \mathrm{L}_s$ of positions from the local area L_s is empty. If s is α-maximal *w.r.t.* both players $\alpha \in \mathbb{B}$, we simply say that it is *maximal* and denote by $\mathrm{St}_{\mathtt{M}} \subseteq \mathrm{St}$ the corresponding subset of states and by $\Game_s \triangleq \Game \setminus \mathsf{dom}\left(\mathsf{r}_s^{(>p_s)}\right)$ 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. By $\mathrm{St}_{\mathtt{S}} \subseteq \mathrm{St}_{\mathtt{M}}$ we denote the set of strongly maximal states. Recall that region R_s of a state s is contained in the quasi α_s-dominion $\mathrm{H}^{\alpha_s,p_s}_s$. We say that s is *open* if the opponent $\overline{\alpha}_s$ can escape from $\mathrm{H}^{\alpha_s,p_s}_s$ starting from R_s using a single move, *i.e.*, if $\mathrm{R}_s \cap \mathsf{esc}^{\overline{\alpha}_s}(\mathrm{H}^{\alpha_s,p_s}_s) \neq \emptyset$. In this case, the opponent may escape from R_s by either moving to the remaining portion of local area $L_s \setminus \mathrm{R}_s$ or to the quasi $\overline{\alpha}_s$-dominion $\mathrm{H}^{\overline{\alpha}_s,p_s}_s$. 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 $\overline{\alpha}_s$-maximal and R_s is α_s-maximal *w.r.t.* L_s. By $\mathrm{St}_{\mathtt{P}} \subseteq \mathrm{St}$ we denote the set of promotable states.

The new priority-promotion-based approach, called *recursive priority promotion* (RPP, for short), is reported in Algorithm 1. The left-hand side shows the main function **sol**, while the right-hand side provides the auxiliary function NextPr and the two procedures Maximise and Promote. The function **sol** assumes the input state s to be maximal, *i.e.*, $s \in \mathrm{St}_{\mathtt{M}}$. At Line 1 it checks if there are still unprocessed positions in the game, namely if the priority of the current state is different from $\bot$. If this is the case, Line 2 maximises the region of the current state, namely $\mathsf{R}_s \triangleq \mathsf{r}^{-1}(p)$, by computing its α_s-attractor, so that the resulting set is α_s-maximal and, therefore, s becomes strongly maximal, *i.e.*, $s \in \mathrm{St}_{\mathtt{S}}$. For convenience, we abbreviate the update of some component in a state s, say component R_s for instance, simply as $\mathsf{R}_s \leftarrow exp$, for some expression exp. Therefore, the instruction at Line 2 updates the state s by replacing the original region R_s with $\mathsf{atr}^{\alpha_s}_{\supset_s}(\mathsf{R}_s)$ within the state. If the resulting state s is closed, it is also promotable, *i.e.*, $s \in \mathrm{St}_{\mathtt{P}}$, being maximal by hypothesis, and, therefore, a promotion can be applied at Lines 4, by means of a call to procedure Promote. If, instead, s is open, which means that the opponent can escape from R_s moving outside the quasi α_s-dominion $\mathrm{H}^{\alpha_s,p_s}_s$, the algorithms proceeds to analyse the part of the game still unprocessed. To do this, we first compute the next state by means of $\mathsf{NextPr}(s)$, which simply identifies the next priority to consider, namely the maximal priority of the unprocessed positions. The resulting state is then given as input to the recursive call at Line 4. 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 ends after analysing all the previously unprocessed positions. As a consequence, either the opponent cannot escape from R_s anymore or it can only move to its own quasi dominion $\mathrm{H}^{\overline{\alpha}_s,p_s}_s$. Line 5 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.*, $s \in \mathrm{St}_{\mathtt{P}}$, and Line 7 promotes the region within the quasi α_s-dominion. If, on the other hand, s is still open, then the opponent can escape to $\mathrm{H}^{\overline{\alpha}_s,p_s}_s$ from some positions in R_s. This means that the current state is not $\overline{\alpha}_s$-maximal and Line 6 fixes this by calling the procedure Maximise. The aim of this function is to reestablish maximality of the quasi dominions $\mathrm{H}^{\alpha_s,p_s}_s$ and $\mathrm{H}^{\overline{\alpha}_s,p_s}_s$ associated with the state s. This is done by attracting positions from the current region $\mathsf{R}_s = \mathsf{L}_s$. The surviving positions in R_s, if any, need not form a quasi α_s-dominion anymore and are set by Maximise to their original priority according to the priority function pr of the game. In any case, when the computation reaches Line 9, whether coming from Line 6, Line 7 or Line 8, the state s is maximal, *i.e.*, $s \in \mathrm{St}_{\mathtt{M}}$, and a second, and final, recursive call is performed on s to process the remaining positions in L_s, if any.

The auxiliary function NextPr generates a new maximal state $\widehat{s} = \mathsf{NextPr}(s) \in \mathrm{St}_{\mathtt{M}}$, starting from a strongly-maximal one $s \in \mathrm{St}_{\mathtt{S}}$. 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 $\bot$.

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

Algorithm 1: RPP Solver

```
   function sol(s: St_M): Rg
1    if p_s ≠ ⊥ then
2      R_s ← atr_{∂_s}^{s}(R_s)
3      if s is open then
4        r_s ← sol(NextPr(s))
5        if s is open then
6          Maximise(s)
         else
7          Promote(s)
       else
8        Promote(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      X ← atr (H_s \ L_s, L_s)
3      q ← min(rng(r_s ↾ (H_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^{‾s}(R_s, r_s)
2    r_s ← r_s[R_s ↦ q]
```

can be attracted by the quasi α-dominions H_s^α, with $\alpha \in \mathbb{B}$. To this end, the procedure computes at Line 2 the α-attractor $\mathsf{atr}^\alpha(\mathrm{H}_s^\alpha \setminus \mathrm{L}_s, \mathrm{L}_s)$, collecting all the position of L_s that player α can force to move into $\mathrm{H}_s^\alpha \setminus \mathrm{L}_s$. The minimum priority q assigned by r to a position in the attracting set is extracted at Line 3. The attracted positions are then assigned priority q in the region function r_s at Line 4. 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, as prescribed at Line 5.

To conclude, the procedure Promote requires a promotable state $s \in \mathrm{St}_{\mathtt{P}}$ 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:

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

where $I \triangleq Mv_{\eth} \cap (\mathsf{esc}^{\overline{\alpha}_s}(\mathsf{R}_s) \times (\mathsf{dom}(\mathsf{r}_s) \setminus \mathsf{R}_s))$ contains all the moves leading outside R_s that the opponent can use to escape. The procedure, then, promotes R_s to q, by assigning at Line 2 the priority q to all the positions of R_s in the region function r_s. Observe that, thanks to the $\overline{\alpha}_s$-maximality of the input state, q is necessarily congruent to α_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 $\mathbf{sol}(\eth) \triangleq (\mathsf{r}^{-1}(\top_0), \mathsf{r}^{-1}(\top_1))$, 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.

Algorithm 3: Parys Solver

function $\mathbf{sol}(s\colon \mathrm{St})\colon 2^{\mathrm{Ps}}$
  if $p_{\mathsf{s}} = \bot \lor b_{0\mathsf{s}} = 0 \lor b_{1\mathsf{s}} = 0$ **then**
    return $(\mathsf{r}_{\mathsf{s}}, \mathsf{u}_{\mathsf{s}})$
  else
    $\mathbf{hsol}(s)$
    $\widehat{s} \leftarrow s$
    $(\mathsf{r}_{\mathsf{s}}, \mathsf{u}_{\mathsf{s}}) \leftarrow \mathbf{sol}(\mathsf{NextPr}(s))$
    $\mathsf{Maximise}(s)$
    if $s \prec \widehat{s}$ **then** $\mathbf{hsol}(s)$
    return $\mathsf{Und}(s)$

procedure $\mathbf{hsol}(s\colon \mathrm{St})$
  repeat
    $\mathsf{R}_{\mathsf{s}} \leftarrow \mathsf{atr}_{\eth_s}^{\ s}(\mathsf{R}_{\mathsf{s}})$
    $\widehat{s} \leftarrow s$
    $(\mathsf{r}_{\mathsf{s}}, \mathsf{u}_{\mathsf{s}}) \leftarrow \mathbf{sol}(\mathsf{Half}(s))$
    $\mathsf{Maximise}(s)$
  until $s \not\prec \widehat{s}$

Auxiliary Functions / Procedures I

function $\mathsf{NextPr}(s\colon \mathrm{St})\colon \mathrm{St}$
  return $((\mathsf{r}_{\mathsf{s}}, p_{\mathsf{s}} - 1), (\mathsf{u}_{\mathsf{s}}, p_{\mathsf{s}}, b_{0\mathsf{s}}, b_{1\mathsf{s}}))$

function $\mathsf{Half}(s\colon \mathrm{St})\colon \mathrm{St}$
  $(b_{0\mathsf{s}}, b_{1\mathsf{s}}) \leftarrow (\left\lfloor \frac{\mathsf{b}_{0s}}{1+\ \ s} \right\rfloor, \left\lfloor \frac{\mathsf{b}_{1s}}{2-\ \ s} \right\rfloor)$
  return $\mathsf{NextPr}(s)$

Auxiliary Functions / Procedures II

procedure $\mathsf{Maximise}(s\colon \mathrm{St})$
  $\mathrm{X} \leftarrow \mathsf{atr}^{\overline{\ }\,s}(\mathrm{H}_{\mathsf{s}}^{\overline{\ }\,s}, \mathrm{L}_{\mathsf{s}})$
  $\mathsf{r}_{\mathsf{s}} \leftarrow \mathsf{r}_{\mathsf{s}} \setminus \mathrm{X}$
  $\mathsf{u}_{\mathsf{s}} \leftarrow \mathsf{u}_{\mathsf{s}}[\mathrm{X} \mapsto p_{\mathsf{s}}]$

function $\mathsf{Und}(s\colon \mathrm{St}) : \mathrm{Rg} \times \mathrm{Pm}$
  $\mathsf{r}_{\mathsf{s}} \leftarrow \mathsf{r}_{\mathsf{s}}^{(>\mathsf{p}_s)}[v \in \mathrm{U}_{\mathsf{s}} \mapsto \mathsf{pr}(v)]$
  $\mathsf{u}_{\mathsf{s}} \leftarrow \mathsf{u}_{\mathsf{s}}[\mathrm{L}_{\mathsf{s}} \mapsto p_{\mathsf{s}} + 1]$
  return $(\mathsf{r}_{\mathsf{s}}, \mathsf{u}_{\mathsf{s}})$

3.2 Parys' Algorithm

In order to obtain a quasi-polynomial time priority-promotion-based solution procedure, we entangle the algorithm of the previous subsection with Parys' idea [48] to suitably truncate the recursion tree. Naturally, cutting some of the recursive calls may prevent us from deciding the winner for some of the positions with certainty. These intermediate results are thus *undetermined* (we use a function u, read 'undetermined', to refer to these results). Parys' contribution was to design the truncation in a way that offers bounded guarantees, namely that these 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. A first step in the integration of Parys' approach with Priority Promotion is to formulate it in the same terms and to use the opportunity to introduce the notation needed when hybridising the approaches. To this end, we assume $\mathrm{U}_s \triangleq \mathsf{u}_s^{-1}(p_s)$ is the set of undetermined positions at a certain stage s of the search. We require that U_s satisfies the following property: it must contain all the $\overline{\alpha}$-dominions of size less than or equal to some given bound b_α and it cannot intersect any α-dominion of size less than or equal to a second given bound $b_{\overline{\alpha}}$.

Just as pure Priority Promotion does not use undetermined positions, Parys' approach does not use regions (beyond the attractor of the nodes with highest priority). Consequently, little happens to the region functions 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.

We only outline the principles here together with slight generalisations of the standard lemmas employed in each step, required later for hybrid approach. In $\mathbf{hsol}$, the attractor of the positions with maximal priority is removed, and the recursive call of **sol** adds all small dominions $\leq \left\lfloor \frac{\mathsf{b}_{\overline{\alpha}_s}}{2} \right\rfloor$ (but no part of any small region ($\leq \mathsf{b}_{\alpha_s}$) of player α) of the remaining subgame to U_s. Lemma 2 provides that, if such an $\overline{\alpha}_s$ dominion was contained in the game before removing R_s then a sub-dominion of it still remains after R_s is removed.

Lemma 1. *If a dominion D for Player α in a game $\eth$ does not intersect with a set S of nodes, then it does not intersect with $\mathsf{atr}^{\overline{\alpha}}_{\eth}(S)$ either.*

For $\alpha \equiv_2 p$, a p-Region is a quasi α-dominion, such that (1) all nodes have priority $\leq q$, and (2) all escape positions have priority p.

Lemma 2. *If the highest priority in a non-empty dominion D for player α is p and it intersects with a q-region R for $q \not\equiv_2 \alpha$ and $q \geq p$, then α's parity (i.e. $\alpha \neq p \mod 2$), then D contains a non-empty sub-dominion that does not intersect with $\mathsf{atr}^{\overline{\alpha}}_{\eth}(R)$.*

These dominions are then closed under attractor by a call of $\mathsf{Maximise}$, until s does not change, and thus until no $\overline{\alpha}_s$ dominion $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$ is left. This guarantee is used in **sol** to ensure that a dominion D of player $\overline{\alpha}_s$ is in U_s at the end of the function: all $\overline{\alpha}_s$ dominions returned in the recursive call of line 5 are $> \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$. Closing them under attractor through a call of $\mathsf{Maximise}$ (line 6) leaves, by Lemma 1, a (possibly empty) $\overline{\alpha}_s$ dominion of size $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$, when **hsol** is called for the second time, and thus fully included in U_s. After Parys' Solver is run (with full precision) on a game with maximal priority p_{max}, $u^{-1}(p_{\mathsf{max}} + 1)$ contains the winning region of player $p_{\mathsf{max}} \mod 2$.

3.3 A Hybrid Algorithm

In our hybrid approach, we have to synthesise the use of regions and the use of undetermined sets. To formalise this intuition, a state for the hybrid algorithm needs to embed quite a bit of additional information *w.r.t.* a simple state of RPP. It obviously contains both a region function r, tracking the quasi dominions already analysed, and the current priority p. It also features an additional promotion function u, used to maintain the set of undetermined positions, which only satisfy the relative guarantees mentioned above, the priority of the caller used in the update of this function, and two numbers b_0 and b_1, representing the two bounds *w.r.t.* which the guarantees are expressed. The initial state, from which the search starts, contains, as is the case in the exponential algorithm, the priority function pr and the maximal priority $\mathsf{pr}(\eth)$. In addition, the two bounds are both set to the number of positions in the game, while the accessory function u is empty. For technical convenience, we set the caller priority to $\top_0$. Finally, the ordering between the states is again defined in terms of the sets of unprocessed positions in the two states.

Definition 6 (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. $\mathrm{S} \subseteq (\mathrm{Rg} \times \mathrm{Pr}_{\perp}) \times (\mathrm{Pm} \times \mathrm{Pr}_{\top} \times \mathbb{N} \times \mathbb{N})$ *is the set of all tuples* $s \triangleq ((\mathsf{r}, p), (\mathsf{u}, c, b_0, b_1))$*, called* hybrid states*, where:*
   a) $\mathsf{dom}(\mathsf{r}) \cap \mathsf{dom}(\mathsf{u}) = \emptyset$, $\mathsf{dom}(\mathsf{r}) \cup \mathsf{dom}(\mathsf{u}) = \mathrm{Ps}$, $\mathsf{dom}\big((\mathsf{r} \cup \mathsf{u})^{(>p) \wedge (<c)}\big) = \emptyset$, $p < c$;

b) $\mathsf{u}^{-1}(\top_1) = \mathsf{dom}\big(\mathsf{u}^{(<p)}\big) = \varnothing$ *and if* $c \neq \top_0$ *then* $\mathsf{u}^{-1}(\top_0) = \varnothing$*;*
c) $\mathsf{dom}(\mathsf{u}) \cap \mathsf{esc}^{\overline{\alpha}}(\mathrm{H}_s^{\alpha,q}) = \emptyset$*, with* $\mathrm{H}_s^{\alpha,q} \triangleq \mathrm{H}_{\mathsf{r}}^{\alpha,q} \cup \mathrm{H}_{\mathsf{u}}^{\overline{\alpha},q}$*, for all* $\alpha \in \mathbb{B}$ *and* $q \geq p$*;*
for every state $s \in \mathrm{St}$*, we set (i)* $\alpha_s \triangleq p \bmod 2$*, (ii)* $\mathrm{H}_s^{\alpha} \triangleq \mathrm{H}_{\mathsf{r}}^{\alpha} \cup \mathrm{H}_{\mathsf{u}}^{\overline{\alpha}}$*, (iii)* $\mathrm{U}_s \triangleq \mathsf{u}^{-1}(p)$*,* *(iv)* $\mathrm{R}_s \triangleq \mathsf{r}^{-1}(p)$*, and (v)* $\mathrm{L}_s \triangleq \mathsf{dom}\big(\mathsf{r}^{(\leq p)}\big)$*;*

2. $s_I \triangleq ((\mathsf{pr}, \mathsf{pr}(\partial)), (\varnothing, \top_0, |\partial|, |\partial|))$ *is the* initial state*;*
3. $s_1 \prec s_2$ *if either* $\mathrm{L}_{s_1} \subset \mathrm{L}_{s_2}$ *or* $\mathrm{L}_{s_1} = \mathrm{L}_{s_2}$ *and* $p_{s_1} < p_{s_2}$*.*

Intuitively, Item 1a ensures that the set of positions in the game is partitioned into two categories: *(i)* those contained in the region function r, which are considered *determined*, in the sense that they belong to known quasi dominions; and *(ii)* those contained in the promotion function u, which are *undetermined*, since they form sets that only satisfy the bounded guarantees. Obviously the priority of the caller has to be higher than the current one and no position can be associated with a priority between those two. Moreover, since positions assigned to the two top pseudo-priorities must be determined, as they form dominions, u cannot refer to those two values, except for the outermost call, when $c = \top_0$. In this case, indeed, any undetermined position is necessarily won by player 1, *i.e.*, $\mathsf{u}^{-1}(\top_0) \subseteq \mathrm{Wn}_1$. Moreover, all positions with priorities lower than the current one are still unprocessed, therefore they cannot be undetermined. Both these requirements are expressed by Item 1b. Finally, we need to ensure that a player cannot immediately escape from a set of undetermined positions, as specified in Item 1c. This property is crucial to maintain, after the update of the promotion function u, the implicit invariant stating that, if a strongly-maximal state is closed, then it is promotable.

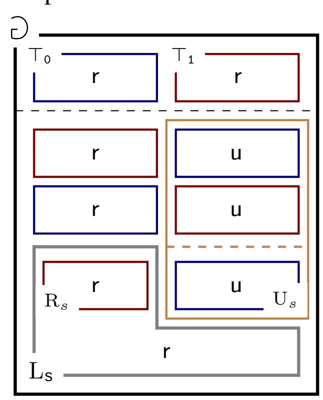

Fig. 2. The structure of a hybrid state.

Figure 2 reports a graphical representation of the structure of a hybrid state. Most of the concepts and notation introduced for the states of RPP have a similar meaning and play a similar role for hybrid states. 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 game, while $\mathrm{H}_s^{0,q}$ and $\mathrm{H}_s^{1,q}$ represent the portions of these sets having a priority, assigned either in r or in u, at least equal to $q \in \mathrm{Pr}$. As opposed to the previous notion of state, however, these sets are not necessarily quasi dominions, since they may include undetermined positions, namely those contained in $\mathrm{H}_u^{0,q}$ and $\mathrm{H}_u^{1,q}$. Only the two subsets $\mathrm{H}_{\mathsf{r}_s}^0$ and $\mathrm{H}_{\mathsf{r}_s}^1$, as well as their relativised versions $\mathrm{H}_{\mathsf{r}_s}^{0,q}$ and $\mathrm{H}_{\mathsf{r}_s}^{1,q}$, are known to be quasi dominions.
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 $\mathrm{H}_s^{\alpha} \setminus \mathrm{L}_s$ 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 $\mathrm{St}_{\mathtt{M}} \subseteq \mathrm{St}$ the set of maximal hybrid states and with $\partial_s \triangleq \partial \setminus \mathsf{dom}\Big(\mathsf{r}_s^{(>p_s)} \cup \mathsf{u}_s\Big)$ the induced subgame over the local area L_s. A maximal hybrid state s is *strongly maximal*, if the current region R_s is α_s-maximal *w.r.t.* L_s and the the quasi $\overline{\alpha}_s$-dominion $\mathrm{H}_s^{\overline{\alpha}_s} \setminus (\mathrm{L}_s \cup \mathrm{U}_s)$ is $\overline{\alpha}_s$-maximal *w.r.t.* $\mathrm{L}_s \cup \mathrm{U}_s$. By $\mathrm{St}_{\mathtt{S}} \subseteq \mathrm{St}_{\mathtt{M}}$ we denote the set of strongly maximal hybrid states. Again, we say that s is

open if $\mathrm{R}_s \cap \mathsf{esc}^{\overline{\alpha}_s}(\mathrm{H}_s^{\alpha_s, p_s}) \neq \emptyset$, and we say that it is *closed*, otherwise. For technical convenience, we always consider a hybrid state with an empty region open. 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. By $\mathrm{St}_{\mathtt{P}} \subseteq \mathrm{St}$ we denote the set of promotable hybrid states.

The *hybrid priority-promotion algorithm* (HPP, for short) reported in Algorithm 6 synthesises the recursive priority-promotion technique of Algorithm 1 and the recursion-tree truncation idea of Algorithm 3 in a single approach. As for the RPP, the main function **sol** assumes the input state s to be maximal, *i.e.*, $s \in \mathrm{St}_{\mathtt{M}}$. Line 1 checks whether (i) there are no unprocessed positions in the game or (ii) one of the two bounds on the guaranties over the undetermined positions has reached threshold zero. If one of these conditions is satisfied, the current region function r_s and the promotion function u_s are returned unmodified at Line 2, as no further progress can be achieved in the current recursive call. Otherwise, similarly 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 via 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). Once these three phases terminate, the information about the undetermined positions still contained in the local area L_s or in the undetermined set U_s is suitably updated by the function Und at Line 10.

To discuss the guarantees and their effects in more detail, let us fix a small dominion D, with $|\mathrm{D}| \leq b_{\overline{\alpha}_s}$. The call to **hsol** at Line 3 modifies in-place the maximal state s given as input into a strongly-maximal one such that $\mathsf{dom}(\mathsf{r}^{(\leq p_s)})$ does no longer contain any tiny dominions of player $\overline{\alpha}_s$ of size $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$. Moreover, $\mathrm{D}' = \mathrm{D} \cap \mathsf{dom}(\mathsf{r}^{(\leq p_s)})$ is a dominion in $\mathsf{dom}(\mathsf{r}^{(\leq p_s)})$, while $\mathrm{D} \setminus \mathrm{D}'$ has been processed and added to $\mathsf{dom}(\mathsf{u}^{(\geq p_s)}) \cup \mathsf{dom}(\mathsf{r}^{(> p_s)})$. After that, at Line 4, the obtained state is locally recorded in order to determine, later on, whether the second phase achieves any progress. The algorithms then proceeds to analyse the remaining part of the game still unprocessed. To do so, the next state computed by $\mathsf{NextPr}(s)$ is given as input to the recursive call at Line 5. Once the call completes, the state is updated with the two new functions returned by the call. At this point, the guarantee that all $\overline{\alpha}_s$ dominions in $\mathsf{dom}(\mathsf{r}^{(\leq p_s)})$ are larger than $\lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$ entails that, if D' is not empty (and thus D completely processed), then a non-empty sub-dominion D'' of D' is part of the call. As $\mathsf{dom}(\mathsf{r}^{(\leq p_s)})$ contains no tiny $\overline{\alpha}_s$ dominion, $|\mathrm{D}''| > \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$, and $\mathrm{D}'' \setminus \mathrm{D}' \leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$.

Depending on whether the state is closed or not, 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 has made some progress in the search, a last call to **hsol** at Line 9 is performed, which again modifies in-place the current state into a strongly-maximal one. As $\mathrm{D}'' \setminus \mathrm{D}' \leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$, it is processed in **hsol**. If no progress occurred, instead, the current state is equal to the one previously returned by the first call to **hsol** and, thus, strongly-maximal. It also entails that D' was empty, and D therefore processed completely in the fist call of **hsol**. In both cases, the state is fed to the function Und, after which the current call terminates.

Algorithm 6: HPP Solver

```
function sol(s: St_M): Rg × Pm
  if p_s = ⊥ ∨ b_0s = 0 ∨ b_1s = 0 then
    return (r_s, u_s)
  else
    hsol(s)
    ŝ ← s
    (r_s, u_s) ← sol(NextPr(s))
    if s is open then
      Maximise(s)
    else
      Promote(s)
    if s ≺ ŝ then hsol(s)
    return Und(s)
```

Algorithm 7: Half-Solver

```
  procedure hsol(s: St_M)
1   repeat
2     R_s ← atr^{α_s}_{⅁_s}(R_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 / Procedures I

```
function NextPr(s: St_S): St_M
  q ← max(rng(r_s^{(<p_s)}))
  return ((r_s, q), (u_s, p_s, b_0s, b_1s))

function Half(s: St_S): St_M
  (b_0s, b_1s) ← (⌊b_0s / (1+α_s)⌋, ⌊b_1s / (2−α_s)⌋)
  return NextPr(s)

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

Auxiliary Functions / Procedures II

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

procedure Maximise(s: St)
  Z ← R_s
  foreach α ∈ 𝔹 do
    X ← atr^α(H_s^α \ L_s, L_s ∪ U_s)
    q ← min(rng((r_s ∪ u_s) ↾ (H_s^α \ L_s)))
    if q ≡_2 α then
      r_s ← r_s[X ↦ q]
      u_s ← u_s \ X
    else
      r_s ← r_s \ X
      u_s ← u_s[X ↦ q]
  if Z ≠ R_s then r_s ← r_s[v ∈ L_s ↦ pr(v)]
```

The procedure **hsol** simply executes the main body of the RPP algorithm by making mutually-recursive calls to the **sol** function (Line 5) with halved precision, until no progress on the search for a dominion can be made. The auxiliary function NextPr generates a new maximal state $\widehat{s} = \mathsf{NextPr}(s) \in \mathrm{St}_{\mathtt{M}}$, starting from the strongly-maximal one $s \in \mathrm{St}_{\mathtt{S}}$ in input. The state $\widehat{s}$ is obtained by first setting the caller priority to the current one p_s and, then, by computing the highest priority q among the unprocessed positions in

$\mathsf{L}_s \setminus \mathsf{R}_s$. The promotion and maximisation procedures, $\mathsf{Promote}$ and $\mathsf{Maximise}$, generalise the corresponding ones associated with RPP. The only difference is that here we need to determine which one, between the region function r and the promotion function u, has to receive the promoted region Rg_s, in case of $\mathsf{Promote}$, or the positions attracted from $\mathrm{L}_s \cup \mathrm{U}_s$, in case of $\mathsf{Maximise}$. As before, $\mathsf{Promote}$ asks for the input state to be promotable, *i.e.*, $s \in \mathrm{St}_{\mathsf{P}}$, while $\mathsf{Maximise}$ does not require any specific property on it.

Similarly to Parys' algorithm, the Half function halves the bound of the opponent player $\overline{\alpha}_s$, leaving the bound of player α_s unchanged. Finally, the Und function, starts from a strongly maximal state $s \in \mathrm{St}_{\mathsf{S}}$ where all small dominions of player $\overline{\alpha}_s$ of size $\leq b_{\overline{\alpha}_s}$ *from the time of the 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, we can then return the respective set 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 $\mathrm{W}_1 = \mathsf{r}^{-1}(\top_1) \cup \mathsf{u}^{-1}(\top_0)$ and $\mathrm{W}_0 = \mathrm{Ps} \setminus \mathrm{W}_1$, *i.e.*, $\mathbf{sol}(\supset) \triangleq (\mathrm{W}_0, \mathrm{W}_1)$, 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.

4 Correctness and Complexity

We now discuss how we can entangle the concepts of Priority Promotion—the transfer of information across the call structure, which makes it so efficient in practice—with the concept of relative guarantees that provides favourable complexity guarantees to Parys' algorithm. Before turning to the principle guarantees provided by the algorithm, we note that the two algorithms from the previous sections, Parys' algorithm and the selected variation of Priority Promotion, can be viewed as variations of our hybrid algorithm. This is particularly easy to see for the exponential Priority Promotion algorithm from Section 3.1: 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 3. The connection to Parys' algorithm is slightly looser, but essentially it replaces $\mathsf{Maximise}$ by closing only U_s under attractor, and skips the promotions (through calling $\mathsf{Promote}$) altogether. Note that these changes would not impact the partial correctness argument, while the remaining parts of the algorithm alone are strong enough to guarantee progress.

4.1 Correctness

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 in all function calls. These properties are not entirely local, and to conveniently reason about the effect of updates, we use $\mathrm{B}_s = \mathrm{H}_s^{\overline{\alpha}} \setminus \mathrm{R}_s = \mathrm{H}_{\mathsf{u}}^{\alpha,p_s} \cup (\mathrm{H}_{\mathsf{r}}^{\overline{\alpha},p_s} \setminus \mathrm{R}_s)$ for the states that are *bad for Player* α_s in that they contain the states in $u_s^{-1}(q)$ for all $q \geq p_s$ with $q \equiv_2 \alpha_s$ and all states in $r_s^{-1}(q)$ for all $q > p_s$ with $q \not\equiv_2 \alpha_s$, and $\mathrm{G}_s = \mathrm{H}_s^{\alpha} \setminus \mathrm{R}_s = \mathrm{H}_{\mathsf{u}}^{\overline{\alpha},p_s} \cup (\mathrm{H}_{\mathsf{r}}^{\alpha,p_s} \setminus \mathrm{R}_s)$ for the states that are *good for Player* α_s in that they contain the states in $u_s^{-1}(q)$ for all $q > p_s$ with $q \not\equiv_2 \alpha_s$ and all states in $r_s^{-1}(q)$ for all $q > p_s$ with $q \equiv_2 \alpha_s$.

We also introduce additional data for each function, namely a set P_s, which stores the local area L_s at the beginning of the call of **sol**, which 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 this initial set, 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 theorem. While the proofs for the auxiliary functions and of the main result will be reported in the extended version, while the inductive proof of the correctness is outlined below.

Theorem 2. *Let* $s \in \mathrm{St}_{\mathtt{M}}$ *be a maximal hybrid state for a parity game* $\supseteq$*, where* $c_s = \mathsf{min}\{\mathsf{dom}(\mathrm{P}_s) \cup \top_0\}$ *and* $b_{0s}, b_{1s} \geq 1$*. Assume* **sol** *is called on* s *and let* $s' \triangleq ((\widehat{\mathsf{r}}, p'), (\widehat{\mathsf{u}}, c', b_{0s}, b_{1s}))$ *be the hybrid state, for* $(\widehat{\mathsf{r}}, \widehat{\mathsf{u}}) \triangleq \mathbf{sol}(s)$*,* $p' \triangleq c_s$*, and some* $c' > c_s$*. The following holds:*

- *if* $c_s \equiv_2 \alpha_s$ *then* $\mathrm{B}_s = \mathrm{B}_{s'}$*, hence,* $\mathrm{B}_{s'}$ *preserves all global guarantees of* B_s*.*
- *if* $c_s \not\equiv_2 \alpha_s$ *then:*
  - G_s *contains all small dominions of player* α_s *of size* $\leq b_{\alpha_s}$ *in* P_s *and intersects with no small* $\overline{\alpha}_s$ *dominions of size* $\leq b_{\overline{\alpha}_s}$ *in* P_s*; and*
  - $\mathrm{B}_{s'} = \mathrm{G}_s$.

Moreover, if **hsol** *is called on* $s \in \mathrm{St}_{\mathtt{M}}$*, then it modifies* s *into a strongly-maximal hybrid state with the following property:* L_s *does not contain a small dominion of player* $\overline{\alpha}_s$ *of size* $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$*, and* R_s *and* B_s *are closed under attractor in* P_s*.*

Proof sketch. We prove this theorem by induction, using the lemmas from the previous section. Establishing the base case for **sol** and maximal priority $\perp$, **hsol** and maximal priority 0, and **sol** and maximal priority 0 is straight forward.

The induction step for **hsol** then establishes that, after return, L_s contains no tiny $\overline{\alpha}_s$ dominion of size $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$, as they would otherwise (using Lemma 2) be found in the last recursive call of **sol**. We also have that R_s is closed under attractor because, since closed states lead to promotion, the resulting state s must be open, so that $\mathsf{Maximise}$ provides closure of R_s under attractor.

Using this, the induction step for **sol** has mainly to show that an initial small dominion D ($|\mathrm{D}| \leq b_{\overline{\alpha}_s}$) of player $\overline{\alpha}_s$ is entirely in B_s when Und is called. Let D' be the intersection of D with L_s. Obviously, D' is a dominion of player $\overline{\alpha}_s$ (Lemma 1). If D' is empty, we are done. Otherwise, D' must have a non-empty sub-dominion D'' that does not intersect with R_s (Lemma 2), and these are added, by inductive hypothesis, to B_s by the full precision call (Line 5). In addition, $|\mathrm{D}''| > \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$ by the return guarantees of **hsol**. The rest of the dominion included in $\mathrm{D}' \setminus \mathrm{D}''$, is then $\leq \lfloor \frac{b_{\overline{\alpha}_s}}{2} \rfloor$ and, by the guarantees of **hsol**, is added to B_s by the second call of **hsol** (note that we have $s' \prec s$).

With the global guarantee that B_s does not intersect with small dominions of size $\leq b_{\alpha_s}$ of player α_s, we have added all small dominions of the players $\overline{\alpha}_s$ and α_s to B_s and $\mathrm{G}_s \cup \mathrm{L}_s$, respectively and, depending on the priority, we can insert either U_s or L_s into $u^-1(c_s)$. □

Correctness is then the special case that we call **sol** with $c_s = \top_0$ and full precision.

Corollary 1. *When* **sol** *is called with the initial state s_I,* i.e. *with $c_s = \top_0$ and full precision $b_\alpha = |\eth|$, for all $\alpha \in \mathbb{B}$, then, after* **sol** *returns, $r^{-1}(\top_1) \cup u^{-1}(\top_0)$ is the winning region of player* 1. □

This is because $r^{-1}(\top_1)$ and $r^{-1}(\top_0)$ contain dominions of the respective players and are closed under attractor by our global guarantees. It is also clear that $u^{-1}(\top_0)$ can only be filled in the final call of Und, such that $u^{\top_0} \cup r^{\top_1}$ contains all dominions (as the bound does not exclude any) of Player 1, but does not intersect with any dominion (as the bound again does not include any) dominion of Player 0. 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 the winning regions of both players: $r^{-1}(\top_0)$ and $r^{-1}(\top_1)$, are computed constructively, and winning strategies are simply the contribution attractor strategies / arbitrary (on their subgame) exits from states with the dominating priority of their region. This is not the case for the remainder of the winning regions, as their calculation is not constructive.

4.2 Complexity

As a hybrid between Priority Promotion and Parys' approach, the algorithm retains both complexity bounds. We show the quasi-polynomial bound, as retaining the quasi-polynomial bound and practical efficiency of Priority Promotion is our main goal. The argument is exactly the same as Parys' (Section 5 of [48]): we use 2 parameters, c for the number of priories, and $l = 2\lfloor \log_2(\mathsf{pos}) \rfloor + 1$, where pos is the number of positions. We estimate the number of times **sol** is called, excluding the trivial calls that return immediately (in line 2), because we run out if priorities or bounds, by $R(h, l)$. If $h = 0$, then we have run out of priorities ($p_s = \bot$), and $R(h, l) = 0$. If $l = 0$, then we have run out of bounds ($b_{0s} = 0$ or $b_{1s} = 0$, with the other value being 1). As argued in Section 5 of [48], we can estimate $R(h, l) \leq n^l \binom{h+l}{l} - 1$. As the cost of all operations are linear in the size of the game, and as l is logarithmic in the number of positions, this provides us with a quasi-polynomial running time.

5 Experimental Evaluation

The practical effectiveness of the solution algorithms presented here, namely RPP and HPP, has been assessed by means of an extensive experimentation on both concrete and synthetic benchmarks. Both the algorithms have been incorporated in Oink [59], a tool written in C++ that collects implementations of several parity game solvers proposed in the literature, including the known quasi-polynomial ones. We shall compare solution times against the quasi-polynomial solvers SSPM [34], QPT [21], and ZLKQ [47], as well as the best exponential solvers, namely the optimised version of the recursive algorithm ZLK from [43] and the original priority promotion NPP [7], whose superiority in practical contexts is widely acknowledged (see, *e.g.*, [52,59]). The benchmarks considered include the games from [36] as well as some additional synthetic randomly generated games, designed to further stress the considered solvers.[3]

5.1 Keiren's Benchmarks

The first set of benchmarks we consider were first proposed in [36] and 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 $O(10^7)$ positions. It includes a number of different verification problems. A first set contains encodings of a variety of communication protocols from [37,14,30,2]: the alternating bit protocol, the positive acknowledgement with retransmission protocol, the bounded 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 [62] and the wait-free handshake register of [31], as well as the classic elevator and towers-of-Hanoi benchmarks from [27]. 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 $O(10^7)$ nodes 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 have been obtained with the tool MLSolver [28]. 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 1 reports the results of the experiments for all the solvers considered in the analysis, divided by class of benchmarks. For each solver, the total completion time, the

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

Benchmarks		Exponential ZLK	NPP	RPP	Quasi Polynomial SSPM	QPT	ZLKQ	**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%**

Table 1. Solution times in seconds on Keiren's benchmarks (1012 games).

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 is used instead. As expected, the exponential solvers perform better on all the classes, with NPP 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 timeouts. Both ZLKQ and HPP, instead, perform relatively well in all the benchmarks, being able to solve all the instances without incurring in timeouts and maintaining a short distance from the exponential solvers performance-wise. ZLKQ 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 ZLKQ and practically matches the performance of the exponential solvers. 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, whose results are reported in the next subsection.

5.2 Randomly generated benchmarks

The objective of this second set of benchmarks is 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 benchmaks.

Figure 3 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. On the left-hand side the number of priorities grows linearly with the positions, *i.e.*, equal to $\frac{n}{4}$, with n the number of positions. On the right-hand side, instead, all the games have 500 priorities. In both cases, the timeout was set to 25 seconds. In these experiments, HPP and ZLKQ are joined by 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 ZLKQ, 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.

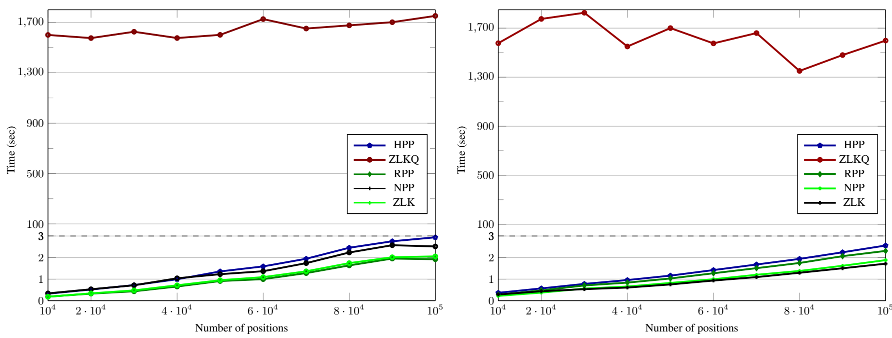

Fig. 3. Results on random games with linear (on the left) and fixed (on the right) number priorities.

What seems to emerge from the experimental analysis is that HPP behaves quite nicely in practice, often 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, unlike 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 quite well for the applicability of the approach in the more challenging practical contexts, such as deciding temporal logic properties or solving reactive synthesis problems, where the number of priorities is typically higher.

Worst Case		NPP		u-ZLK		HPP	
Index	Nodes	Time	Iterations	Time	Iterations	Time	Iterations
10	88	0.00	11267	0.00	1295	0.00	69
15	170	0.18	524294	0.00	10175	0.00	177
20	278	7.70	22020104	0.03	133631	0.00	339
25	410	-	-	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 the robust worst case family [5].

5.3 Robust Worst Case Family

Figure 3 shows that, for simple games, HPP is the only quasi-polynomial algorithm that behaves very much like the exponential solvers. For complex games, instead, it is well known that even the most efficient solvers in practice, *i.e.* u-ZLK [25,60,8] and NPP [4,60,8], take exponential time, while HPP has a quasi-polynomial worst case complexity. To show this behaviour, we have evaluated these three solvers on the robust worst case family [5]. The results are reported in Table 2. Clearly the complex infrastructure required by the HPP can pay off in terms of running time.

Note that in the experiments of Table 1 and Figure 3 we used the optimised implementation of Zielonka's algorithm, the ZLK, while in this benchmark we used the original version, namely u-ZLK. The reason for this choice is that the improvement targets the structure of the hard examples, without changing the complexity of the algorithm. Thus, the 'hard examples' would need to be updated in the usual arms race. Such an update is possible, but adjusting the benchmarks does not add to the insight.

6 Discussion

We have synthesised two generalisations of the classic recursive algorithm: the quasi-polynomial recursion scheme of Parys, which is 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 in Section 5 suggests that the algorithm lives up to this promise. The salable random benchmarks suggest that our solver behaves very much like the fastest solver classes available—Zielonka and Priority Promotion. There is a slight overhead to pay for the additional data structure, but this is merely a small linear factor of some 50%. While this is well within the range that depends on the quality of implementation, we assume that the slight overhead of our more expressive data structure shows. Most notably, the behaviour differs significantly from Parys' algorithm, which behaves – unsurprisingly – quite nicely when the number of priorities is so small that it never runs out of bound. Just like our algorithm, it simply behaves like its parent algorithm, Zielonka, in that case. The data also shows quite nicely at what cost the more efficient treatment of pathological examples come in Parys' approach—ironically, its weakness in practice are games with a high number of priorities. While our hybrid algorithm runs out of bounds in the majority of these cases, too, it proves to be quite resilient: the presence of entries to the u function has no similar effect on our approach.

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". The work was supported by EPSRC grant EP/P020909/1.

References

1. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.
2. K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson. A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. *CACM*, 12(5):260–261, 1969.
3. M. Benerecetti, D. Dell'Erba, and F. Mogavero. Improving Priority Promotion for Parity Games. In *HVC'16*, LNCS 10028, pages 1–17. Springer, 2016.
4. M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Parity Games via Priority Promotion. In *CAV'16*, LNCS 9780 (Part II), pages 270–290. Springer, 2016.
5. M. Benerecetti, D. Dell'Erba, and F. Mogavero. Robust Exponential Worst Cases for Divide-et-Impera Algorithms for Parity Games. In *GANDALF'17*, EPTCS 256, pages 121–135, 2017.
6. M. Benerecetti, D. Dell'Erba, and F. Mogavero. A Delayed Promotion Policy for Parity Games. *IC*, 262(2):221–240, 2018.
7. M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Parity Games via Priority Promotion. *FMSD*, 52(2):193–226, 2018.
8. M. Benerecetti, D. Dell'Erba, and F. Mogavero. Robust Worst Cases for Parity Games Algorithms. *IC*, 272:104501:1–31, 2020.
9. D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-Width and Parity Games. In *STACS'06*, LNCS 3884, pages 524–536. Springer, 2006.
10. H. Björklund and S.G. Vorobyov. A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean-Payoff Games. *DAM*, 155(2):210–229, 2007.
11. Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theoretical Computer Science*, 310(1–3):365–378, 2004.
12. A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1–2):237–255, 1997.
13. C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding Parity Games in Quasipolynomial Time. In *STOC'17*, pages 252–263. ACM, 2017.
14. V.G. Cerf and R.E. Kahn. A Protocol for Packet Network Intercommunication. *TC*, 22(5):637–648, 1974.
15. Laure Daviaud, Marcin Jurdzinski, and K. Thejaswini. The strahler number of a parity game. In *Automata, Languages and Programming, 47th International Colloquium, ICALP, Proceedings, Part II*, 2020.
16. Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proc. LICS*, pages 279–290. IEEE Computer Society Press, June 2001.
17. E.A. Emerson and C.S. Jutla. Tree Automata, muCalculus, and Determinacy. In *FOCS'91*, pages 368–377. IEEECS, 1991.
18. E.A. Emerson, C.S. Jutla, and A.P. Sistla. On Model-Checking for Fragments of muCalculus. In *CAV'93*, LNCS 697, pages 385–396. Springer, 1993.
19. E.A. Emerson and C.-L. Lei. Efficient Model Checking in Fragments of the Propositional muCalculus. In *LICS'86*, pages 267–278. IEEECS, 1986.
20. J. Fearnley. Non-Oblivious Strategy Improvement. In *LPAR'10*, LNCS 6355, pages 212–230. Springer, 2010.
21. J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An Ordered Approach to Solving Parity Games in Quasi Polynomial Time and Quasi Linear Space. In *SPIN'17*, pages 112–121. ACM, 2017.
22. J. Fearnley and S. Schewe. Time and Space Results for Parity Games with Bounded Tree and DAG Width. *LMCS*, 9(2):1–31, 2013.

23. John Fearnley, Sanjay Jain, Bart de Keijzer, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *International Journal on Software Tools for Technology Transfer*, 21(3):325–349, 2019.
24. O. Friedmann. An Exponential Lower Bound for the Latest Deterministic Strategy Iteration Algorithms. *LMCS*, 7(3):1–42, 2011.
25. O. Friedmann. Recursive Algorithm for Parity Games Requires Exponential Time. *RAIROTIA*, 45(4):449–457, 2011.
26. O. Friedmann. A Superpolynomial Lower Bound for Strategy Iteration Based on Snare Memorization. *DAM*, 161(10-11):1317–1337, 2013.
27. O. Friedmann and M. Lange. Solving Parity Games in Practice. In *ATVA'09*, LNCS 5799, pages 182–196. Springer, 2009.
28. O. Friedmann and M. Lange. A Solver for Modal Fixpoint Logics. *ENTCS*, 262:99–111, 2010.
29. Oliver Friedmann. A subexponential lower bound for zadeh's pivoting rule for solving linear programs and games. In *Proc. of IPCO*, LNCS, pages 192–206, 2011.
30. J.F. Groote and J. van de Pol. A Bounded Retransmission Protocol for Large Data Packets. In *AMST'96*, LNCS 1101, pages 536–550. Springer, 1996.
31. W.H. Hesselink. Invariants for the Construction of a Handshake Register. *IPL*, 68(4):173–177, 1998.
32. M. Jurdziński. Deciding the Winner in Parity Games is in UP $\cap$ co-UP. *IPL*, 68(3):119–124, 1998.
33. M. Jurdziński. Small Progress Measures for Solving Parity Games. In *STACS'00*, LNCS 1770, pages 290–301. Springer, 2000.
34. M. Jurdziński and R. Lazic. Succinct Progress Measures for Solving Parity Games. In *LICS'17*, pages 1–9. ACM, 2017.
35. M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SJM*, 38(4):1519–1532, 2008.
36. J.J.A. Keiren. Benchmarks for Parity Games. In *FSEN'15*, LNCS 9392, pages 127–142. Springer, 2015.
37. C.P.J. Koymans and J.C. Mulder. A Modular Approach to Protocol Verification Using Process Algebra. In *Applications of Process Algebra.*, pages 261–306. CUP, 1990.
38. D. Kozen. Results on the Propositional muCalculus. *TCS*, 27(3):333–354, 1983.
39. O. Kupferman and M.Y. Vardi. Module Checking Revisited. In *CAV'97*, LNCS 1254, pages 36–47. Springer, 1997.
40. Martin Lange. Solving parity games by a reduction to SAT. In *Proc. Int. Workshop on Games in Design and Verification*, 2005.
41. K. Lehtinen. A Modal mu Perspective on Solving Parity Games in Quasi-Polynomial Time. In *LICS'18*, pages 639–648. ACM & IEEECS, 2018.
42. K. Lehtinen, S. Schewe, and D. Wojtczak. Improving the Complexity of Parys' Recursive Algorithm. Technical report, arXiv, 2019.
43. Y. Liu, Z. D., and C. Tian. An Improved Recursive Algorithm for Parity Games. In *TASE'14*, pages 154–161. IEEECS, 2014.
44. Walter Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Inf. Comput.*, 117(1):151–155, 1995.
45. R. McNaughton. Infinite Games Played on Finite Graphs. *APAL*, 65:149–184, 1993.
46. J. Obdržálek. Fast mu-calculus model checking when tree-width is bounded. In *Proc. CAV*, pages 80–92. Springer-Verlag, 2003.
47. P. Parys. Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time. Conference under submission.

48. P. Parys. Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time. In *MFCS'19*, LIPIcs 138, pages 10:1–13. Leibniz-Zentrum fuer Informatik, 2019.
49. Paweł Parys. Parity games: Another view on lehtinen's algorithm. In *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
50. N. Piterman. From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata. In *LICS'06*, pages 255–264. IEEECS, 2006.
51. Anuj Puri. *Theory of hybrid systems and discrete event systems*. PhD thesis, Computer Science Department, University of California, Berkeley, 1995.
52. L. Sanchez, W. Wesselink, and T. A. C. Willemse. A Comparison of BDD-Based Parity Game Solvers. In *GANDALF'18*, EPTCS 277, pages 103–117, 2018.
53. S. Schewe. An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games. In *CSL'08*, LNCS 5213, pages 369–384. Springer, 2008.
54. S. Schewe. Solving Parity Games in Big Steps. *JCSS*, 84:243–262, 2017.
55. S. Schewe and B. Finkbeiner. Satisfiability and Finite Model Property for the Alternating-Time muCalculus. In *CSL'06*, LNCS 6247, pages 591–605. Springer, 2006.
56. S. Schewe, A. Trivedi, and T. Varghese. Symmetric Strategy Improvement. In *ICALP'15*, LNCS 9135, pages 388–400. Springer, 2015.
57. Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *Proceedings of the Twelfth International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2009), 22–29 March, York, England, UK*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer-Verlag, 2009.
58. Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2006), 12–14 July, Venice, Italy*, volume 4407 of *Lecture Notes in Computer Science*, pages 127–142. Springer-Verlag, 2006.
59. T. van Dijk. Oink: an Implementation and Evaluation of Modern Parity Game Solvers. In *TACAS'18*, LNCS 10805, pages 291–308. Springer, 2018.
60. T. van Dijk. A Parity Game Tale of Two Counters. In *GANDALF'19*, EPTCS 305, pages 107–122, 2019.
61. M.Y. Vardi. Reasoning about The Past with Two-Way Automata. In *ICALP'98*, LNCS 1443, pages 628–641. Springer, 1998.
62. R. Veldema, R.F.H. Hofman, R. Bhoedjang, C.J.H. Jacobs, and H.E. Bal. Source-Level Global Optimizations for Fine-Grain Distributed Shared Memory Systems. In *PPOPP'01*, pages 83–92. ACM, 2001.
63. J. Vöge and M. Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *CAV'00*, LNCS 1855, pages 202–215. Springer, 2000.
64. T. Wilke. Alternating Tree Automata, Parity Games, and Modal muCalculus. *BBMS*, 8(2):359–391, 2001.
65. W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *TCS*, 200(1-2):135–183, 1998.
66. U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *TCS*, 158(1-2):343–359, 1996.