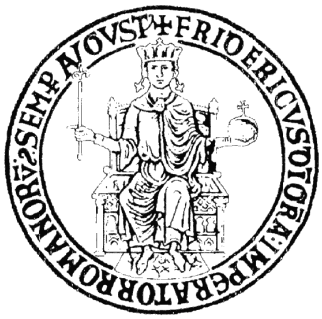
Solving Infinite Games on Graphs via Quasi Dominions



Daniele Dell'Erba

Università Degli Studi Di Napoli Federico II
Dipartimento di Matematica e Applicazioni

Dottorato di Ricerca in Scienze Matematiche e Informatiche

Supervisors

Prof. Dr. Massimo Benerecetti

Dr. Fabio Mogavero

External Reviewers

Prof. Dr. Salvatore La Torre

Prof. Dr. Sven Schewe

Abstract

In this thesis we present a new approach to efficiently solve two types of infinite-duration games on graphs, namely *parity games* and *mean-payoff games*, which can be used, in the context of formal verification, for the model checking and synthesis of hardware and software systems. Our main aim is to improve the performances of the corresponding solution algorithms in order to support their effective use in practice.

We start with the analysis of one of the principal algorithms for parity games developed by Zielonka, called the *Recursive Algorithm*. This study lead to the discovery of an infinite family of games whose structure makes them very difficult to solve even by improved versions of the considered algorithm. A more robust version of this family is proved to be challenging for most of other parity algorithms known in the literature as well. We then present a novel technique based on the notion of *quasi dominions* that is able to defeat all of worst case families that were known in the literature. This technique, called *priority promotion* (PP, for short), is the focal point of a *divide et impera* algorithm that solves a game by looking for subsets of the winning regions by exploiting a relaxed notion of dominion which is precisely a quasi dominion. The obtained procedure exhibits the best known space complexity up to date, a result in line with the purpose of this thesis, *i.e.*, making a scalable solution algorithm in practice. The experiments we conducted show that the approach is extremely effective on both concrete and synthetic problems. Due to the subsequent discovery of an exponential-time worst case for the PP algorithm, we also revised the proposed approach as a framework general enough to be instantiated with different game-decomposition techniques and solution policies. The resulting refinements of PP method significantly mitigate the exponential behaviors exhibited by the vanilla approach. A first refinement consists in a finer policy promotion, called *delayed promotion*, that delays some of the promotions that are the cause of the recomputation of discarded results that were already computed before. A second one, instead, called *region recovery*, simply tries to recover these results that would, instead, be needlessly discarded. The effectiveness of each refinement is strongly supported by experimental results. Every presented algorithm, indeed, outperforms those previously proposed.

We finally investigate, with success, the possible application of the quasi-dominion based technique to solve mean-payoff games, by presenting an algorithm that matches the best known time complexity of the problem. Also in this case, we significantly improve on the practical application of solution algorithms, by proving the existence of a family of games on which the new algorithm performs arbitrarily better than the classic approaches.

Contents

Chapter 1

Introduction

The more reliable a system is, the more useful it is. The only way to completely ensure that a system is correct is to formally verify it, a practice that is, unfortunately, very expensive to carry out. For this reason, currently, most of the validation process rely on testing, a much cheaper technique at the price of showing the presence of errors, but never ensuring their absence. On one hand the existence of safety critical systems, which are software or hardware whose failure can have serious consequences, such as put human lives, the environment or high value goods in dangerous, requires the application of formal verification. On the other hand, the usually enormous size and the complexity of these systems, discourage this practice.

To establish the reliability of a system, it is necessary to ensure that it correctly implements the designed specification. In the context of verification, the system is represented by a model and the specification by a logic formula, and one has to check whether the model satisfies the formula, a notion derived from mathematical logic. This process, called *model-checking problem*, concretely consists at solving a particular type of game on graphs, whose structure depends on both the model and the type of employed logic. The *synthesis problem* is also related to the reliability of a system. In this case, however, the system is not first built and then verified, but directly synthesized from the specification, and therefore, it is correct by construction. This all but trivial process can also be reduced to a solution of a game. It is therefore quite evident that devising algorithms for efficiently solving games in practice is a crucial endeavor towards enabling a wide application of these techniques at an industrial level. In the present thesis, we try to pursue this goal by presenting new algorithms for the solution of two types of games on graphs: *parity games* and *mean-payoff games*.

Parity games [Mos91] are perfect-information two-player turn-based games of infinite duration, usually

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

Parity games have been extensively studied in the attempt to find efficient solutions to the problem of determining the winner. From a complexity theoretic perspective, this decision problem lies in NPTime ∩ CoNPTime [EJS93,EJS01], since these games are *memoryless determined* [Mos91,EJ91,Mar75,Mar85]. It has been even proved to belong to UPTime ∩ CoUPTime [Jur98], a status shared with the *factorization problem* [FK92a, FK92b, AKS04]. They are the simplest class of games in a wider family with similar complexities and containing, *e.g.*, *mean payoff games* [EM79, GKK88], *discounted payoff games* [ZP96], and *simple stochastic games* [Con92]. In fact, polynomial time reductions exist from parity games to the latter ones. However, despite being the most likely class among those games to admit a polynomial-time solution, the answer to the question whether such a solution exists still eludes the research community.

The effort devoted to provide efficient solutions stems primarily form the fact that many problems in formal verification and synthesis can be reformulated in terms of solving parity games. Emerson, Jutla, and Sistla [EJS93, EJS01] have shown that computing winning strategies for these games is linear-time equivalent to solving the modal μCalculus model checking problem [EL86]. Parity games also play a crucial role in automata theory [Mos84, EJ91, KV98], where, for instance, they can be applied to solve the emptiness of the nondeterministic alternating tree automata [KV98]. These automata, in turn, can be used to solve the satisfiability and model checking problems for expressive logics, such as the modal [Wil01] and alternating [SF06, AHK02] μCalculus, ATL* [AHK02, Sch08b], Strategy Logic [CHP10, MMV10, MMPV12, MMPV14], Substructure Temporal Logic [BMM13, BMM15], and fixed-point extensions of guarded first-order logics [BG01, BG04].

Previous solutions mainly fall into two families: those that solve a game by first decomposing it into smaller subgames, and those that proceed in a global fashion and approach the game in its entirety. To the first family belongs the *divide et impera* solution originally proposed by McNaughton [McN93] for Muller games and adapted to parity games by Zielonka [Zie98]. More recent improvements to that recursive algorithm have been proposed by Jurdziński, Paterson, and Zwick [JPZ06, JPZ08] and by Schewe [Sch07]. Both approaches rely on finding suitably closed dominions, which can then be removed from a game to reduce the size of the subgames to be recursively solved. To the second family belongs

the procedure proposed by Jurdziński [Jur00], which exploits the connection between the notions of progress measures [KK91] and winning strategies. In this approach, an initial measure function on the entire game is iteratively updated until a fixpoint is reached. At that point, a progress measure is obtained that induces a winning strategy for one of the two players. An alternative approach was proposed by Vöge and Jurdziński [VJ00], which directly builds a winning strategy for one of the two players, by iteratively improving an initial non-winning strategy. This technique was later optimized by Schewe [Sch08a]. A recent breakthrough [CJK$^+$17] by Calude *et al.* proposes a succinct reduction from parity to reachability games based on a clever encoding of the sequences of priorities a player finds along a play. This allows for a mere quasi-polynomial blow up in the size of the underlying graph and sets the basis of the fixed-parameter tractability *w.r.t.* the number of priorities. The approach has been then considerably refined in [FJS$^+$17], where these encodings are modeled as progress measures. A similar technique is also used in [JL17], while two different approaches have been presented in [Leh18] and in [Par19, LSW19]. Despite the theoretical relevance of this new idea, preliminary experiments conducted in this paper suggest that the practical impact of the result may not match the theoretical one. Indeed, most of the exponential algorithms mentioned above outperform, often by orders of magnitude, the current implementations of the quasi-polynomial ones, which do not scale beyond a few hundred positions. This evaluation is consistent with the fact that the new techniques essentially amount to clever and succinct encodings embedded within a brute force search, which makes matching quasi-polynomial worst cases quite easy to find. As far as space consumption is concerned, we have different and, in some cases, incomparable behaviors. The small progress measure procedure of [Jur00] requires $\mathrm{O}(k \cdot n \cdot \log n)$ space, with n the number of positions in the game and k the number of its priorities. On the other hand, it is $\mathrm{O}\big(n^2\big)$ for the optimized strategy improvement method of [Sch08a]. Due to their inherent recursive nature, the algorithms of the first family require $\mathrm{O}(m \cdot n)$ memory, where m denotes the number of edges of the underlying graph. This bound could, in principle, be reduced to $\mathrm{O}\big(n^2\big)$, by representing subgames implicitly through their sets of positions. The lowest space requirements, however, are those of the more recent quasi-polynomial algorithm described in [JL17], which only requires $\mathrm{O}(n \cdot \log n \cdot \log k)$ additional space. All these bounds do not seem to be amenable to further improvements, as they appear to be intrinsic to the corresponding solution techniques. Polynomial time solutions are only known for restricted versions of the problem, where one among tree-width [Obd03,FL11,FS12], dag-width [BDHK06], clique-width [Obd07] and entanglement [BGKR12] of the underlying graph is bounded.

We propose a new algorithm, called *priority promotion*, for the solution of this decision problem, based on the idea of promoting vertexes to higher priorities during the search for winning regions. The proposed approach has nice computational properties, exhibiting the best space complexity among the

currently known solutions. Experimental results on both random games and benchmark families show that the technique is also very effective in practice. The underlying framework is general enough to accommodate different instantiations of the solution procedure, whose correctness is ensured by the nature of the space itself. We then propose two different instantiations, called *delayed promotion* and *region recovery*, that try to reduce the possible exponential behaviors exhibited by the original method in the worst case. The resulting procedures often outperforms both the state-of-the-art solvers and the original priority promotion approach.

Mean-payoff games [EM79] are, instead, infinite-duration perfect-information two-player games played on weighted directed graphs, each of whose vertexes is controlled by one of the two players. The game starts at an arbitrary vertex and, during its evolution, each player can take moves at the vertexes it controls, by choosing one of the outgoing edges. The moves selected by the two players induce an infinite sequence of vertices, called play. The payoff of any prefix of a play is the sum of the weights of its edges. A play is winning if it satisfies the game objective, called *mean-payoff objective*, which requires that the limit of the *mean payoff*, taken over the prefixes lengths, never falls below a given *threshold* ν.

Mean-payoff games have been first introduced and studied by Ehrenfeucht and Mycielski in [EM79], who showed that positional strategies suffice to obtain the optimal value. A slightly generalized version was also considered by Gurvich *et al.* in [GKK88]. Positional determinacy entails that the decision problem for these games lies in NPTime $\cap$ CoNPTime [ZP96], and it was later shown to belong to UPTime $\cap$ CoUPTime [Jur98], being UPTime the class of unambiguous non-deterministic polynomial time. This result gives the problem a rather peculiar complexity status, shared by very few other problems, such as integer factorization [FK92a], [AKS04] and parity games [Jur98]. Despite various attempts [GKK88, ZP96, Pis99, DG06, BV07], no polynomial-time algorithm for the mean-payoff game problems is known so far.

A different formulation of the game objective allows to define another class of quantitative games, known as *energy games*. The *energy objective* requires that, given an initial value c, called *credit*, the sum of c and the *payoff* of every prefix of the play never falls below 0. These games, however, are tightly connected to mean-payoff games, as the two type of games have been proved to be log-space equivalent [BFL$^+$08]. They are also related to other more complex forms of quantitative games. In particular, polynomial-time reductions [Jur98] exist from these games to *discounted payoff* [ZP96] and *simple stochastic games* [Con92].

Recently, a fair amount of work in formal verification has been directed to consider, besides correctness properties of computational systems, also quantitative specifications, in order to express performance measures and resource requirements, such as quality of service, bandwidth and power consumption

and, more generally, bounded resources. Mean-payoff and energy games also have important practical applications in system verification and synthesis. In [CHJ09] the authors show how quantitative aspects, interpreted as penalties and rewards associated to the system choices, allow for expressing optimality requirements encoded as mean-payoff objectives for the automatic synthesis of systems that also satisfy parity objectives. With similar application contexts in mind, [BCHK11] and [BBFR13] further contribute to that effort, by providing complexity results and practical solutions for the verification and automatic synthesis of reactive systems from quantitative specifications expressed in linear time temporal logic extended with mean-payoff and energy objectives. Further applications to temporal networks have been studied in [CR15] and [CPR17]. Consequently, efficient algorithms to solve mean-payoff games become essential ingredients to tackle these problems in practice.

Several algorithms have been devised in the past for the solution of the decision problem for mean-payoff games, which asks whether there exists a strategy for one of the players that grants the mean-payoff objective. The very first deterministic algorithm was proposed in [ZP96], where it is shown that the problem can be solved with $\mathrm{O}\big(n^3 \cdot m \cdot W\big)$ arithmetic operations, with n and m the number of positions and moves, respectively, and W the maximal absolute weight in the game. A strategy improvement approach, based on iteratively adjusting an initial strategy for one player until a winning strategy is obtained, is presented in [Sch08a], which has an exponential upper bound in the number of positions. however, this upper bound does not depends on the maximal weight. The algorithm by Lifshits and Pavlov [LP07], which runs in time $\mathrm{O}(n \cdot m \cdot 2^n \cdot \log_2 W)$, computes the "potential" of each game position, which corresponds to the initial credit that the player needs in order to win the game from that position. Algorithms based on the solution of linear feasibility problems over the tropical semiring have been also provided in [ABG14b, ABG14a, ABGJ15]. The best known deterministic algorithm to date, which requires $\mathrm{O}(n \cdot m \cdot W)$ arithmetic operations, was proposed by Brim *et al.* [BCD$^+$11]. They adapt to energy and mean-payoff games the notion of progress measures [Kla91], as applied to parity games in [Jur00]. The approach was further developed in [CR17] to obtain the same complexity bound for the optimal strategy synthesis problem. A strategy-improvement refinement of this technique has been introduced in [BC12]. Finally, Bjork *et al.* [BSV04] proposed a randomized strategy-improvement based algorithm running in time $\mathsf{min}\{\mathrm{O}\big(n^2 \cdot m \cdot W\big), 2^{\mathrm{O}\left(\sqrt{n} \cdot \log n\right)}\}$.

We propose a novel algorithm for the solution of *mean-payoff games* that merges together two seemingly unrelated concepts introduced in the context of parity games, *small progress measures* and *quasi dominions*. We show that the integration of the two notions can be highly beneficial and significantly speeds up convergence to the problem solution. Experiments show that the resulting algorithm performs orders of magnitude better than the asymptotically-best solution algorithm currently known, without

sacrificing on the worst-case complexity.

The main content of this thesis is organized in two parts. The first one simply provides the formal introduction and the necessary context to the problem, all notions related to both parity and mean-payoff games, and a description of the current state-of-art algorithms. The second part, instead, contains all the original contributions that were are part of the following publications in collaboration with M. Benerecetti and F. Mogavero:

- Robust Worst Cases for Parity Games Algorithms [BDM17, BDM19a];
- Solving parity games via priority promotion [BDM16c, BDM18b];
- A delayed promotion policy for parity games [BDM16a, BDM18a];
- Improving Priority Promotion for Parity Games [BDM16b];
- Solving Mean-Payoff Games via Quasi Dominions [BDM19b].

Part I

Turn-Base Infinite Games on Graphs

Chapter 2

Games on Graphs

In this chapter we present parity and mean-payoff games, which are the subject of the thesis. We shall give all the needed definitions to formalize the games along with examples showing how this particular games are played. This first part decribes the general syntax of games, while Section 2.1 and 2.2 focus on parity and mean-payoff games, respectively.

Given a partial function $\mathsf{f} : \mathrm{A} \rightharpoonup \mathrm{B}$, by $\mathsf{dom}(\mathsf{f}) \subseteq \mathrm{A}$ and $\mathsf{rng}(\mathsf{f}) \subseteq \mathrm{B}$ we indicate the domain and range of f, respectively. The *completion operator* $\uplus$ takes a partial function f and another partial function $\mathsf{g} : \mathrm{A} \rightharpoonup \mathrm{B}$, returns the partial function $\mathsf{f} \uplus \mathsf{g} \triangleq (\mathsf{f} \setminus \mathsf{dom}(\mathsf{g})) \cup \mathsf{g} : \mathrm{A} \rightharpoonup \mathrm{B}$, which is equal to g on its domain and assumes the same values of f on the remaining part of A.

We describe game arenas, the structure that underlies games over graphs. A graph is composed of a set of vertices Ps and a set of moves Mv. We consider turn-based games, therefore in each vertex only one player can choose a move. The player are usually denoted as 0 and 1. Accordingly, the set of vertices of the game Ps is partitioned in two sets, Ps^{0} and Ps^{1}, in such way that player $\wp$, with $\wp \in \{0, 1\}$, is the owner of the vertices in $\mathrm{Ps}^{\wp}$. Similarly, we shall use the symbol $\overline{\wp}$ to denote the opponent player to $\wp$. Therefore, $\overline{\wp} = 1$ when $\wp = 0$, and *vice versa*. Moreover, the games we are interested in have infinite duration, therefore we shall only consider *total* graphs, namely graphs whose vertices have at least one leaving move (there are no sink vertices). When the token is on a vertex in $\mathrm{Ps}^{\wp}$, player $\wp$ can take a move. The moves selected by the players induce an infinite sequence of vertices, called play, corresponding on infinite path in the graph.

Definition 2.0.1 (Game arena)**.** *An two player total graph, called* arena*, is a tuple* $\mathcal{A} = \langle \mathrm{Ps}^{0}, \mathrm{Ps}^{1}, Mv \rangle$*, where:*

- $\mathrm{Ps} \triangleq \mathrm{Ps}^{0} \cup \mathrm{Ps}^{1}$ *is the set of vertices with* $\mathrm{Ps}^{0} \cap \mathrm{Ps}^{1} = \emptyset$*;*
- $Mv \subseteq \mathrm{Ps} \times \mathrm{Ps}$ *is a left-total relation describing all possible moves, where for all* $u \in \mathrm{Ps}$ *there exists* $v \in \mathrm{Ps}$ *such that* $(u, v) \in Mv$*.*

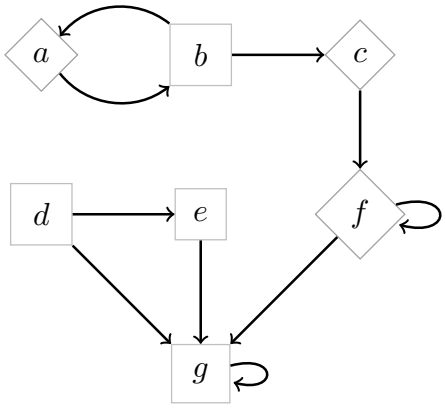


Figure 2.1: A game arena.

Figure 2.1 shows an example of directed arena, where squared shaped vertices belong to Player 1, diamond shaped ones belong to Player 0.

A *path* in $\mathrm{V} \subseteq \mathrm{Ps}$ is a finite or infinite sequence $\pi \in \mathrm{Pth}(\mathrm{V})$ of vertices in V compatible with the move relation, *i.e.*, $(\pi_i, \pi_{i+1}) \in Mv$, for all $i \in [0, |\pi| - 1[$. For a finite path π, with $\mathsf{lst}(\pi)$ we denote the last vertex of π.

Players choose moves according to some $\wp$-strategy, a function that maps each finite history of a play π such that $\mathsf{lst}(\pi) \in \mathrm{Ps}^{\wp}$ to a successor of $\mathsf{lst}(\pi)$. For some games, in particular for those we are interested in, a simpler notion of strategy suffices, where the choice of the next move only depends on the current vertex. Such strategies are called memoryless (or positional), and are encoded as follows.

Definition 2.0.2 (Positional Strategy)**.** *A positional* strategy *for player* $\wp \in \{0, 1\}$ *on* $\mathrm{V} \subseteq \mathrm{Ps}$ *is a partial function* $\sigma_{\wp} \in \mathrm{Str}^{\wp}(\mathrm{V}) \subseteq (\mathrm{V} \cap \mathrm{Ps}^{\wp}) \rightharpoonup \mathrm{V}$*, mapping each* $\wp$*-vertex* $v \in \mathsf{dom}(\sigma_{\wp})$ *to vertex* $\sigma_{\wp}(v)$ *compatible with the move relation,* i.e.*,* $(v, \sigma_{\wp}(v)) \in Mv$*.*

With $\mathrm{Str}^{\wp}(\mathrm{V})$ we denote the set of all $\wp$-strategies on V. Once a player adopts some strategy, not necessarily all the plays are still feasible. The only feasible ones are those which conform with the choices stipulated by the strategy at the vertices of that player. Such plays are called *consistent* with that strategy.

Definition 2.0.3 (Consistent play)**.** *A* Consistent Play *in* $\mathrm{V} \subseteq \mathrm{Ps}$ *from a vertex* $v \in \mathrm{V}$ 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 vertex $v \in \mathrm{V}$ 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 $\mathsf{play}((\sigma_0, \sigma_1), v)$.

The $\wp$-predecessor of V, in symbols $\mathsf{pre}^\wp(\mathrm{V}) \triangleq \{v \in \mathrm{Ps}^\wp : Mv(v) \cap \mathrm{V} \neq \emptyset\} \cup \{v \in \mathrm{Ps}^{\overline{\wp}} : Mv(v) \subseteq \mathrm{V}\}$, collects the vertices from which player $\wp$ can force the game to reach some vertex in V with a single move. The $\wp$-attractor $\mathsf{atr}^\wp(\mathrm{V})$ generalizes the notion of $\wp$-predecessor $\mathsf{pre}^\wp(\mathrm{V})$ to an arbitrary number of moves, and corresponds to the least fix-point of that operator. When $\mathrm{V} = \mathsf{atr}^\wp(\mathrm{V})$, we say that V is $\wp$-maximal. Intuitively, V is $\wp$-maximal if player $\wp$ cannot force any vertex outside V to enter the set. For such a V, the set of vertices of the subgame $\mathcal{Э} \setminus \mathrm{V}$ is precisely $\mathrm{Ps} \setminus \mathrm{V}$.

The set $\mathsf{esc}^\wp(\mathrm{V}) \triangleq \mathsf{pre}^\wp(\mathrm{Ps} \setminus \mathrm{V}) \cap \mathrm{V}$, called the $\wp$-*escape* of V, contains the vertices in V from which $\wp$ can leave V in one move. The dual notion of $\wp$-*interior*, defined as $\mathsf{int}^\wp(\mathrm{V}) \triangleq (\mathrm{V} \cap \mathrm{Ps}^\wp) \setminus \mathsf{esc}^\wp(\mathrm{V})$, contains, instead, the $\wp$-vertices from which $\wp$ cannot escape with a single move, while the notion of $\wp$-*stay*, defined as $\mathsf{stay}^\wp(\mathrm{V}) \triangleq (\mathrm{V} \cap \mathrm{Ps}^\wp) \setminus \mathsf{esc}^{\overline{\wp}}(\mathrm{V})$, denotes the $\wp$-vertices from which $\wp$ has a move to remain in V. Observe that all the operators and sets described above actually depend on the specific game $\mathcal{Э}$ they are applied in. However, we shall only add $\mathcal{Э}$ as subscript of an operator, *e.g.*, $\mathsf{esc}^\wp_{\mathcal{Э}}(\mathrm{V})$, when the game is not clear from the context.

In the game of Figure 2.1 the set $\mathrm{A} = \{e, f\}$ corresponds to the 0-predecessor of vertex g, in symbols $\mathsf{pre}^0(\{g\})$, since Player 0 can reach g from f in a single move, while Player 1 is forced to reach it from e. The set $\{c, d, e, f\}$, instead, is the 0-attractor for the same set, that is $\mathsf{atr}^0(\{g\})$. If we consider the set $BSet = \mathrm{V} \setminus \{g\}$, its 0-escape is equal to $\mathrm{A} = \mathsf{esc}^0(\mathrm{B})$, while the remaing set of vertices $\{a, b, c, d\} = \mathsf{int}^0(\mathrm{B})$ corresponds to the dual notion of 0-interior. Finally, the 0-stay set of $BSet$ is $\{a, c, f\}$, since all the vertices owned by 0 have a successor in $BSet$.

In order to formally define the notion of game, we first need to introduce the notion of winning play. Given a play $\pi = v_0 v_1 v_2 \ldots$, we denote with Π^ω the set of infinite plays of an arena. For each player $\wp$, a subset $\mathrm{P}_\wp \subseteq \Pi^\omega$ is chosen as the set of winning plays. For adversarial games, such as those we consider in this thesis, the winning plays P_0 and P_1 for the two players are disjoint. As a consequence, we shall only consider games where $\mathrm{P}_1 = \Pi^\omega \setminus \mathrm{P}_0$, *i.e.*, where all the plays not winning for player 0 are winning for player 1.

Definition 2.0.4 (Game)**.** *A two player game is a pair* $\mathcal{Э} = \langle \mathcal{A}, \mathrm{P} \rangle$*, where* $\mathcal{A} = \langle \mathrm{Ps}^0, \mathrm{Ps}^1, E \rangle$ *is a game arena and* P *is the set of winning plays for player 0.*

A game $\mathcal{G}' = \langle \mathcal{A}', \mathrm{P} \rangle$ is called a subgame of $\mathcal{G} = \langle \mathcal{A}, \mathrm{P} \rangle$ if $\mathcal{A}'$ is a game arena which is also a subgraph of $\mathcal{A}$. Moreover by $\mathcal{G} \setminus \mathrm{V}$ we denote the maximal subgame of $\mathcal{G}$ with set of vertices Ps' contained in $\mathrm{Ps} \setminus \mathrm{V}$ and move relation Mv' equal to the restriction of Mv to Ps'. Finally, from this point, we shall refer to the vertices of the game arena as game positions.

Among all the possible strategies for a player, we are interested in those whose consistent plays are winning, regardless of the strategy employed by the opponent player. A strategy $\sigma_\wp$ is a winning strategy for player $\wp$ over the set $\mathrm{V} \subseteq \mathrm{Ps}$ if every play starting from a position in V and consistent with $\sigma_\wp$ is winning for $\wp$. Note that the notion of winning strategy can be equivalently expressed by the following condition: $\sigma_\wp$ is a winning strategy for player $\wp$ over V if and only if for all $v \in \mathrm{V}$ and for all strategy $\sigma_{\overline{\wp}}$ of the opponent $\overline{\wp}$, the outcome play $\mathsf{play}((\sigma_\wp, \sigma_{\overline{\wp}}), v)$ is winning for $\wp$.

A set of positions $\mathrm{V} \subseteq \mathrm{Ps}$ is an $\wp$-*dominion*, with $\wp \in \{0, 1\}$, if there exists an $\wp$-strategy $\sigma_\wp \in \mathrm{Str}^\wp(\mathrm{V})$ such that, for all $\overline{\wp}$-strategies $\sigma_{\overline{\wp}} \in \mathrm{Str}^{\overline{\wp}}(\mathrm{V})$ and positions $v \in \mathrm{V}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ belongs to $\mathrm{P}_\wp$. In other words, $\sigma_\wp$ only induces on V winning plays for $\wp$.

The solution of a game consists of a unique pair of disjoint subsets Wn_0 and Wn_1 of Ps. The set $\mathrm{Wn}_\wp$, with $\wp \in \{0, 1\}$, contains all the positions starting from which player $\wp$ can win, and, therefore the positions for which $\wp$ has a winning strategy.

It is easy to see that the $\wp$-winning set $\mathrm{Wn}_\wp$ corresponds to the maximal $\wp$-*dominion* in the game.

We can now introduce the two class of games which are the subjects of the thesis: *parity games* and *mean-payoff games*.

2.1 Parity Games

Definition 2.1.1 (Parity game)**.** *A* parity game *is a tuple* $\mathcal{G} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathcal{P}$*, where:*

- $\mathcal{A}$ *is an arena;*
- $\mathrm{Pr} \subset \mathbb{N}$ *is a finite set of priorities;*
- $\mathsf{pr} : \mathrm{Ps} \to \mathrm{Pr}$ *is a* priority function *assigning a priority to each position.*

The priority function can be naturally extended to games and paths as follows. For a game $\mathcal{G}$ we set $\mathsf{pr}(\mathcal{G}) \triangleq \mathsf{max}_{v \in \mathrm{Ps}}\, \mathsf{pr}(v)$ and 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. Moreover, Ps_0 (*resp.*, Ps_1) denotes the set of positions with even priority (*resp.*, odd priority). Usually the players are renamed as *Even* and *Odd* (*resp.*, 0 and 1), we shall keep the 0 and 1 notation.

A winning strategy in a parity game for player $\wp$ is an $\wp$-strategy $\sigma_\wp \in \mathrm{Str}^\wp(\mathrm{V})$ such that, for all $\overline{\wp}$-strategies $\sigma_{\overline{\wp}} \in \mathrm{Str}^{\overline{\wp}}(\mathrm{V})$ and positions $v \in \mathrm{V}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ is infinite and $\mathsf{pr}(\pi) \equiv_2 \wp$. In other words, $\sigma_\wp$ is winning if it only induces on V infinite plays whose maximal priority visited infinitely often has parity $\wp$. The maximal set of position within Player $\wp$ has a winning strategy and from which its opponent cannot escape is called *$\wp$-winning region*.

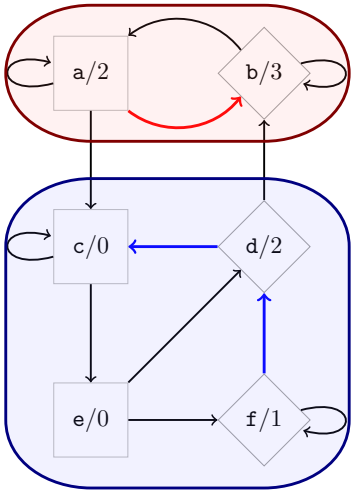


Figure 2.2: A Parity Game.

In Figure 2.2, positions a and b represent the 1-winning region that is highlighted in red, since Player 1 can force to see infinitely often the priority 3 of position b, using the strategy highlighted in red. On the contrary, the set $\{c, d, e, f\}$, highlighted in blue, constitute the winning region for 0, since he can force to see either priority 2 infinitely often or only priority 0 from one point onward, using the strategy highlighted in blue.

We finally cite two important results about parity games, that are the determinacy theorem and the existance of memoriless strategies. The first proposition, states that every position of the game is winning for one of the two players.

Proposition 2.1.1 ([EJ91]). *Parity game are determined, hence for every position $v \in \mathrm{Ps}$, either Player 0 or Player 1 has a winning strategy. Consequently, v belongs to a winning set $\mathrm{Wn}_\wp$ with $\wp \in \{0, 1\}$.*

The second proposition states that parity games admit memoryless strategies.

Proposition 2.1.2 ([Zie98]). *Given a parity game $\mathcal{D} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle$, for each $\wp$ with $\wp \in \{0, 1\}$, there exists a winning memoryless strategy $\sigma_\wp$ for $\wp$, which is winning from each $v \in \mathrm{Wn}_\wp$.*

To conclude the introduction on parity games, we report the complexity class these game belong.

Proposition 2.1.3 ([Jur98]). *Deciding the winner in Parity Games is in* UPTIME ∩ COUPTIME.

In [Jur98], the authors first prove that the problem of deciding the winner in mean-payoff games belong to UPTIME ∩ COUPTIME and than the existence a polynomial time reduction from parity games to mean payoff games. The composition of this two statements implies that also deciding the winner in Parity games belong to UPTIME ∩ COUPTIME.

2.2 Mean-Payoff Games

Definition 2.2.1 (Mean-Payoff game). *A* mean-payoff game *(*MPG*, for short) is a tuple* $\mathfrak{G} = \langle \mathcal{A}, \mathrm{Wg}, \mathsf{wg} \rangle$*, where:*

- $\mathcal{A}$ *is an arena;*
- $\mathrm{Wg} \subset \mathbb{Z}$ *is a finite set of integer weights;*
- $\mathsf{wg}\colon \mathrm{Ps} \to \mathrm{Wg}$ *is a* weight function *assigning a weight to each position.*

With Ps^+ (*resp.*, Ps^-) we denote the set of positive-weight positions (*resp.*, non-positive-weight positions). For convenience, we shall refer to non-positive weights as negative weights. Notice that this definition of MPG is equivalent to the classic formulation in which the weights label the moves, instead.

The weight function naturally extends to paths, by setting $\mathsf{wg}(\pi) \triangleq \sum_{i=0}^{|\pi|-1} \mathsf{wg}(\pi_i)$. The goal of player 0 (*resp.*, 1) is to maximize (*resp.*, minimize) $\mathsf{v}(\pi) \triangleq \liminf_{i\to\infty} \frac{1}{i} \cdot \mathsf{wg}(\pi_{\leq i})$, where $\pi_{\leq i}$ is the prefix up to index i.

Given a threshold ν, a winning strategy for player 0 is a 0-strategy $\sigma_0 \in \mathrm{Str}_0(\mathrm{V})$ such that, for all 1-strategies $\sigma_1 \in \mathrm{Str}_1(\mathrm{V})$ and positions $v \in \mathrm{V}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ satisfies $\mathsf{v}(\pi) > \nu$. In other words, a strategy is winning for 0 (*resp.*, 1) if only induces on V infinite plays whose mean payoff average is higher than ν (*resp.*, at most ν).

The pair of winning regions $(\mathrm{Wn}_0, \mathrm{Wn}_1)$ forms a ν-mean partition. Assuming ν integer, the ν-mean partition problem is equivalent to the 0-mean partition one, as we can subtract ν from the weights of all positions. As a consequence, the MPG decision problem can be equivalently restated as deciding whether player 0 (*resp.*, 1) has a strategy to enforce $\liminf_{i\to\infty} \frac{1}{i} \cdot \mathsf{wg}(\pi_{\leq i}) > 0$ (*resp.*, $\liminf_{i\to\infty} \frac{1}{i} \cdot \mathsf{wg}(\pi_{\leq i}) \leq 0$), for all the resulting plays π.

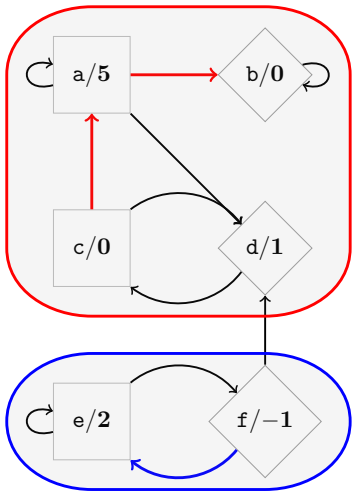


Figure 2.3: A Mean-Payoff Game.

In Figure 2.3, the set of positions $\{a, b, c, d\}$ represent the 1-winning region that is highlighted in red, since Player 1 can force every play to reach the sink b using the strategy highlighted in red and, therefore, obtain a mean payoff zero. On the contrary, e and f, constitute the winning region for 0, that is highlighted in blue, since using the strategy highlighted in blue he can force a play with positive mean payoff.

MPG enjoy the same properties as parity games of determinacy and support of memoryless strategies.

Proposition 2.2.1 ([EM79]). MPG *are determined so for every position* $v \in \mathrm{Ps}$*, either Player* 0 *or* 1 *has a winning strategy. Consequently,* v *belongs to a winning set* $\mathrm{Wn}_\wp$ *with* $\wp \in \{0, 1\}$.

Proposition 2.2.2 ([EM79]). *Given a* MPG $\mathcal{D} = \langle \mathcal{A}, \mathrm{Wg}, \mathsf{wg} \rangle$*, for each* $\wp$ *with* $\wp \in \{0, 1\}$*, there exists a winning memoryless strategy* $\sigma_\wp$ *for* $\wp$*, which is winning from each* $v \in \mathrm{Wn}_\wp$.

We finally report the complexity class of the decision problem for MPG.

Proposition 2.2.3 ([Jur98]). *Deciding the winner in Mean Payoff games is in* UPTIME $\cap$ COUPTIME.

Chapter 3

Parity Algorithms

In this chapter, we describe the most interesting algorithms presented in the literature for solving parity games, which are the result of years of research, improvements, and studies. This overview contains five historical exponential procedures and a description of six more recent quasi-polynomial approaches. The algorithms adopt different solving techniques like decomposition of games, merging of partial solutions or improvement of tentative strategies. They also provide complexity bounds for both space and time, which are useful to compare against.

The first classical algorithm that we present is the recursive procedure by Zielonka [Zie98], followed by the small progress measure approach (SPM, for short) by Jurdziński [Jur00] and strategy improvement by Jurdziński and Vöge [VJ00]. Two of improvements of the recursive algorithm, dominion decomposition [JPZ06] and big step [Sch07], close the set of historical algorithms.

Since the 2017 breakthrough, several quasi-polynomial algorithms have been proposed. Most of them, precisely [CJK+17, FJS+17, JL17, Leh18], reduce to the separation automata technique [CDF+19], while a different approach is used in the quasi-polynomial version of the recursive algorithm by Parys [Par19], which has been further refined in [LSW19].

A final remark is due about the quasi-polynomial algorithms. Experimental evaluations [vD18] show that these algorithm are, at least in the current form, not competitive compared to the exponential ones, both on memory consumption and solution time. This lack of efficiency in practice is due to the fact that every separation automata approach is essentially a quasi-polynomial brute-force search algorithm and, therefore, it is always slower than a classic algorithm that is exponential in its worst case but "practically" polynomial in the average. Furthermore, also the quasi-polynomial improvement of Zielonka's algorithm is very inefficient, as pointed out in [Par19].

3.1 The Recursive Algorithm

The Recursive Algorithm of Zielonka [Zie98] is a specific instantiation for parity games of the algorithm of McNaughton [McN93]. Its procedure, reported in Algorithm 1, contains two recursive calls on different subgames. The presence of these two recursive calls can, in the worst case, lead to an exponential number of recursions.

The game is solved by decomposing it into subgames, until the empty game is obtained, *i.e.* the base case is reached. The game $\eth$ is analyzed by processing priorities starting from the highest one in the game. At each recursive call, a subgame $\eth'$ is generated by removing the $\wp$-attractor A of the set of positions in the current game $\eth$ with the highest priority, where $\wp$ is the player congruent to the current highest priority. If $\eth'$ is completely won by $\wp$, then also the whole $\eth$ is won by $\wp$, since the winning strategy for $\wp$ will be to reach A, where the maximum priority visited infinitely often by all plays is congruent to $\wp$. Any other play that does not reach A is also winning for $\wp$, since $\eth'$ is won by $\wp$.

If, however, $\eth'$ is not completely won by $\wp$, there exist $W'_{\overline{\wp}}$, which is winning for $\overline{\wp}$ in $\eth$, since $\wp$ can not force the positions in $W'_{\overline{\wp}}$ to reach A, the portion of $\eth$ missing in $\eth'$. Since player $\overline{\wp}$ can force positions in A to reach $W'_{\overline{\wp}}$, the $\overline{\wp}$-attractor B on $W'_{\overline{\wp}}$ is obtained to compute the subgame $\eth'' = \eth' \setminus B$, which is then solved by the second recursive call.

A more detailed description of this algorithm is contained in Chapter 5.3. Recently, Parys improved this algorithm by providing a quasi-polynomial version of it as explained in Section 3.7.

Algorithm 1: The Recursive Algorithm of Zielonka.

signature $Solve : \mathcal{P} \rightarrow_{\supset} (2^{\mathrm{Ps}_{\supset}} \times 2^{\mathrm{Ps}_{\supset}})$

function $Solve(\supset)$

1 **if** $\mathrm{Ps}_{\supset} = \emptyset$ **then**

2 **return** $(\emptyset, \emptyset)$

else

3 $p \leftarrow \mathsf{max}(\mathrm{Pr}_{\supset})$

4 $\wp \leftarrow n \bmod 2$

5 $\mathrm{N} \leftarrow \mathsf{pr}_{\supset}^{-1}(n)$

6 $\mathrm{A} \leftarrow \mathsf{atr}_{\supset}^{\wp}(\mathrm{N})$

7 $(\mathrm{W}_{\wp}', \mathrm{W}_{\overline{\wp}}') \leftarrow Solve(\supset \setminus \mathrm{A})$

8 **if** $\mathrm{W}_{\overline{\wp}} = \emptyset$ **then**

9 $(\mathrm{W}_{\wp}, \mathrm{W}_{\overline{\wp}}) \leftarrow (\mathrm{Ps}_{\supset}, \emptyset)$

10 **return** $(\mathrm{W}_{\wp}, \mathrm{W}_{\overline{\wp}})$

else

11 $\mathrm{B} \leftarrow \mathsf{atr}_{\supset}^{\overline{\wp}}(\mathrm{W}_{\overline{\wp}}')$

12 $(\mathrm{W}_{\wp}'', \mathrm{W}_{\overline{\wp}}'') \leftarrow Solve(\supset \setminus \mathrm{B})$

13 $(\mathrm{W}_{\wp}, \mathrm{W}_{\overline{\wp}}) \leftarrow (\mathrm{W}_{\wp}'', \mathrm{W}_{\overline{\wp}}'' \cup \mathrm{B})$

14 **return** $(\mathrm{W}_{\wp}, \mathrm{W}_{\overline{\wp}})$

3.2 Small Progress Measure

This approach, presented in [Jur00], is based on the notion of progress measure function. Given a parity game with d colours, the progress measure function ϱ assigns to each position $v \in \mathrm{Ps}$ a tuple $(c_1, c_2, \ldots, c_d)$ of d colours. A lexicographic order is defined over these tuples with the top value $\top$. Given two tuples $x = (x_1, x_2, \ldots, x_d)$, $y = (y_1, y_2, \ldots, y_d)$ and a colour p, the relation $x >_p y$ holds if the lexicographic order applied to the tuples truncated to the first p colours holds. For example $(1,2,3) >_2 (1,1,4)$ but $(1,2,3) \not>_3 (1,1,4)$. The aim of function ϱ is to assign to each position a tuple, according the following condition for each move $(v, w) \in Mv$:

- if $\lambda(v) \equiv_2 0$ then $\varrho(v) \geq_{\lambda(v)} \varrho(w)$

- otherwise $\varrho(v) >_{\lambda(v)} \varrho(w)$

Given a strategy $\sigma_\wp$ for player $\wp$, we denote with $\Im_{\sigma_\wp}$ the game obtained from $\Im$ by removing all the moves $(v, v') \in Mv$ such that $v \in V_\wp$ and $\sigma_\wp(v) \neq v'$, essentially $\Im_{\sigma_\wp}$ is the subgame of $\Im$ containing all and only the plays consistent with $\sigma_\wp$. We also define an *odd cycle* (*resp.*, even cycle) in a game as the cycles whose maximum position colour is odd (*resp.*, even). In [Jur00] the following two main results have been proved. First, a strategy $\sigma_\wp$ for player $\wp$ is winning if and only if there are no $\overline{\wp}$-cycles in the subgame $\Im_{\sigma_\wp}$. Second, a game $\Im$ admits a progress measure if and only if there exist a strategy $\sigma_\wp$ such that the subgame $\Im_{\sigma_\wp}$ has no $\overline{\wp}$-cycles. As a consequence, a progress measure for player $\wp$ exists if and only if player $\wp$ has a winning strategy.

The solution proposed, then, reduces to computing such a progress measure for player 0. At the beginning of the computation, all the tuples are null, *i.e.*, all the values are equals to 0. For each position v, the algorithm raises the value of the tuple associated to it by means of a lifting function. The lifting of a position v is performed according to the following rule: if the parity of $\lambda(v)$ is odd, then the maximal tuple among those associated with its successors is assigned to v, else the minimal one of its successors is chosen. Positions connected by odd cycles fall into a lifting loop that rises the value of the tuples until the top value $\top$ is reached. When the algorithm terminates, *i.e.* no lifting is possible, positions with defined values of ϱ are winning for player 0. While those with value $\top$ are winning for player 1. Unfortunately, it is possible to create very simple games where there exist positions that need to be lifted an exponential number of times. For instance, it suffice to add to any game a sink position with even priority. This algorithm has time complexity $O\left(d \cdot e \cdot \left(\frac{n}{d}\right)^{d/2}\right)$ and space complexity $O(d \cdot e \cdot \log n)$ [Jur00].

Seventeen years later the development of this algorithm a succinct quasi-polinomial version of it has been presented as described in Section 3.6.

3.3 Strategy Improvement

This approach, presented in [VJ00], works with strategies. The idea is based on a pre-order $\sqsubseteq$ over the strategies of one of the players, called $\wp$. The maximum element $\sigma'_\wp$ of this pre-order, *i.e.* the only element such that, for all $\sigma_\wp$, $\sigma_\wp \sqsubseteq \sigma'_\wp$, is the winning strategy for $\wp$ on the whole winning set $W_\wp$.

Starting form an arbitrary strategy $\sigma_\wp$, the algorithm improves this strategy using an improvement function $improve : \mathrm{Str}^\wp \to \mathrm{Str}^\wp$, such that if $\sigma_\wp$ is not winning, then $\sigma_\wp \sqsubset improve(\sigma_\wp) \not\sqsubseteq \sigma_\wp$, *i.e.* the current strategy has been improved.

Given the current strategy $\sigma_\wp$, the algorithm computes a counter strategy $\sigma_{\overline{\wp}}$ for the opponent $\overline{\wp}$.

The algorithm assign to each position v the value $\mathsf{max}(\mathit{Inf}(\lambda(\pi_{\sigma_\wp,\sigma_{\overline{\wp}}}(v)))$. An improvement is possible if and only if player $\wp$ can modify its strategy $\sigma_\wp$, by choosing a different move for some position, and the evaluation of $\pi_{\sigma'_\wp,\sigma_{\overline{\wp}}}(v)$ is better than $\pi_{\sigma_\wp,\sigma_{\overline{\wp}}}(v)$.

The improvement operation described above is, then, iteratively applied to the initial strategy until a fixpoint is reached.

The cost of a single strategy improvement step is $O(n \cdot m)$, while the maximum number of improvement needed is $O\left(\frac{n}{d}\right)^d$, with d the number of priorities, n the number of positions, and m the number of moves. In [Fri09] the possibility of a chain with an exponential number of improvements is provided. In fact, a family of parity games can be defined such that a binary counter can be encoded within the strategy improvement mechanism. The time complexity for this algorithm is $O(2^e \cdot n \cdot e)$, the space complexity is $O(n^2 + n \cdot \log d + e)$.

A more efficient strategy improvement algorithm has been presented in [Sch08a], with $O(n(\frac{n+d}{d})^{d-i}))$ time complexity where $i = 1 - d \mod 2$. This algorithm is also suitable for mean-payoff games and, therefore, is described in Section 4.2. For this approach no quasi-polynomial version have been developed.

3.4 Dominion Decomposition

The dominion decomposition algorithm [JPZ06] is an evolution of the original Recursive Algorithm. The name dominion decomposition, comes from the notion of *dominion*. An α-dominion is a set of positions for which there exist a strategy for α such that all the consistent plays are winning and contained in the dominion. The winning set for $\wp$ is itself an $\wp$-dominion, but it can contain several smaller $\wp$-dominions. The algorithm proceed as follow. At every iteration it tries to compute an $\wp$-dominions of $l = \sqrt{2 \cdot n}$ size. If such dominion is found it is removed from the game together with its attractor and the residual subgame is processed. If, on the other hand, no $\wp$-dominion is found, the algorithm behaves similarly to the Zielonka algorithm except for the fact that the recursive calls of the original Zielonka's algorithm are replaced with calls to the dominion decomposition algorithm. The search for an $\wp$-dominion is performed by an exhaustive generation of all the possible subsets of the chosen dominion size l. The total time for this algorithm is $n^{O(\sqrt{n})}$, while the space is $O(e \cdot n)$.

3.5 Big Step

For a long time the lowest time complexity known has been the one of Big Step [Sch07]. Similarly to dominion decomposition it is based on the idea of findings dominions to reduce the computation times of

the Recursive Algorithm. However it differs from dominion decomposition in the way $\wp$-dominion are computed. In fact instead of performing an exhaustive search in the space of the possible dominions, it tries to directly compute an $\wp$-dominion by using a restricted version of the progress measure algorithm described above. The restriction of progress measure consist to limit the number of lifting, given a progress measure f, for each tuple $c = (c_1, c_2, \ldots, c_d)$ with d number of priorities, $\sum_{i=0}^{d} f(c_i) \leq u$. The parameter u is set to value $\sqrt[3]{d \cdot n^2}$ that optimize the ratio between the dominions and the game.

Basically, it uses the progress measure approach as a partial solver to find dominions and the dominion decomposition approach on the solved sets to solve the game in larger steps.

With the above mentioned value of u, the resulting time complexity is $O\left(e \cdot n^{d/3}\right)$, while the space complexity is $O(n \cdot (e + d \cdot \log n))$.

3.6 Separation Automata Approach

In this section, we give a quick glance to the *separation automata* approach [BC18] on which several quasi-polynomial algorithms intrinsically rely, in particular the first quasi-polynomial solution for parity games [CJK$^+$17], its efficient version with succinct witness measures [FJS$^+$17], the quasi polynomial improvement of small progress measure [JL17], and the register index approach [Leh18]. The link between all this techniques has been established in [CDF$^+$19], where it is shown that the above algorithms essentially construct a separation automata of quasi polynomial size, even if it is not explicitly stated. Intuitively, the parity problem is reduced to a reachability problem (or, in some cases, to the dual problem of safety), indeed, a reachability game can be obtained by the product of the parity game $\eth$ and a separator automata $\mathcal{A}$ defined as follows. Assume an encoding from plays to words. The pair of sets of words Wn_0 and Wn_1, obtained encoding all the winning and the losing plays for Player 0, forms a partition of all the possible plays of the game. Moreover, the set of words $\mathrm{Wn} = \mathrm{Wn}_0 \cup \mathrm{Wn}_1$ is a language. Therefore, we can build an automata $\mathcal{A}$ that recognize Wn_0 and reject Wn_1. In other words, $\mathcal{A}$ separates all the words encoding plays that are won by 0 from the words obtained from losing plays. In [CDF$^+$19] it is also shown that these automata share a common pattern, precisely each state can be mapped as a leaf of a universal tree. The authors provides different way to build such a tree and, so, the corresponding separation automata. As a consequence of this very important result, we have that the time complexity of every algorithm based on the separation automata approach is lower bounded by the size of universal trees, that unfortunately is at least quasi polynomial, as it is not possible to construct a separation automata with a polynomial number of states.

3.7 Quasi Polynomial Recursive

Twenty years after the development of the Recursive Algorithm [Zie98], Parys presented an interesting quasi-polynomial improvement of it [Par19]. Let us consider a parity game $\mathfrak{D}$ where h is the highest priority in $\mathfrak{D}$, and assume that h is even. The main idea of this improvement is that the k-th recursive call, on a subgame with maximal priority h, returns the set of positions where Player 1 can win visiting a position with priority h at most k times. At the beginning the algorithm assumes that all the position of priority h are winning for Player 0. Then, the first recursive call returns the set $\mathrm{Wn}_1^{(0)}$ of positions where 1 wins without visiting a position with priority h. At this point, the Recursive Algorithm removes the 1-attractor of $\mathrm{Wn}_1^{(0)}$ from $\mathfrak{D}$ obtaining $\mathfrak{D}$' and performs a new recursive call. Note that some position of priority h could have been attracted to $\mathrm{Wn}_1^{(0)}$, therefore the assumption that the h-positions are winning for 0 is no longer valid. Assume the maximal priority in $\mathfrak{D}$' is again h. The second recursive call now returns the set $\mathrm{Wn}_1^{(1)}$ of positions where 1 wins although he visit once a position of priority h. Observe that $\mathrm{Wn}_1^{(0)}$ and $\mathrm{Wn}_1^{(1)}$ are disjoint, since $\mathrm{Wn}_1^{(1)}$ belongs to $\mathfrak{D}$' that has been obtained removing $\mathrm{Wn}_1^{(0)}$ from $\mathfrak{D}$. At the end of the process, after the k-th recursive call, we obtain a partition of k disjoint sets of the winning set of Player 1. Clearly the maximal size of each $\mathrm{Wn}_1^{(i)}$ is n, and moreover, only one of these sets can be larger than n/2 positions. This clever observation allows Parys to introduce a precision bound on the size of the winning set for Player 1 (*resp.* Player 0) when the maximal priority in the current subgame is even (*resp.* odd). As a consequence, the bound avoids the algorithm to perform an exponential number of recursive calls. Indeed, it has $n^{(O(\log n))}$ quasi polynomial time complexity and $O(n \cdot C)$ quadratic space complexity, where n is the number of positions and C the maximal priority. The above modification is so thin to not affect the simplicity of the original approach. But unfortunately, the experimental evaluation on its implementation showed a significant gap between the performance of this version and the original one. On the bright side, this algorithm do not share the same quasi-polynomial worst case of the separation automata approaches. Moreover, it has been recently refined in [LSW19], with a time complexity of $2^{(O(\log n \cdot \log C))}$ and a polynomial time complexity for games with a logarithmic number of priorities.

Chapter 4

Mean-Payoff Algorithms

This chapter describes three interesting algorithms presented in the literature for solving MPGs. In the last decade several approaches have been proposed, both exponential and pseudopolynomial (*i.e.*, polynomial in the size of the game n, but exponential in the binary representation of the maximal absolute weight W). They also differ for the solving technique adopted, some of them are deterministics, others randomized. Moreover, not all the algorithms have the same *power*, indeed MPGs offer multiple problems to solve, such as the value problem (compute the mean payoff for each position), the decision problem (decide for each position whether its mean payoff is strictly positive), and the synthesis of optimal strategies (compute the best winning strategy for each player). We focus on deterministic pseudopolynomial algorithms able to solve the decision problem. In particular the very first solution procedure devised in 1996 [ZP96], the optimal strategy improvement [Sch08a] and the Small Energy Progress Measure [BCD$^+$11] that exhibits the best time complexity to date.

4.1 Value Iteration Algorithm

The Value Iteration Algorithm [ZP96] is the first algorithm developed for MPGs. It works with rational weights, however solve a MPG with a rational weight function is PTime reducible to solve a MPG with integer weights. In this approach a value function μ assigns to each position an approximation of the mean payoff of the plays starting from that position. The procedure is based on a recursive relation that updates the value function. For that procedure, the authors provided two bounds of iteration steps. After the first bound the algorithm solves the decision problem, after the second one, the values became more accurate and it solves also the value problem. This relation is a lifting function that can be described as

follows:

$$\mu(v) \triangleq \begin{cases} \mathsf{max}\{\mu(u) + v : u \in Mv(v)\}, & \text{if } v \in \mathrm{Ps}_0; \\ \mathsf{min}\{\mu(u) + v : u \in Mv(v)\}, & \text{otherwise.} \end{cases}$$

At the beginning μ starts from value zero, that is $\mu(v) = 0$ for each $v \in \mathrm{Ps}$. At each iteration the value of every position is lifted adding the value of an adjacent position to the weight of the position itself. After $4n^2 \cdot W$ steps the value of every position won by Player 0 is strictly positive, and therefore the decision problem is solved, while after $2n^3 \cdot W$ steps the the value function reaches a fix point, providing a solution for the value problem. The algorithm requires $\mathrm{O}\big(n^3 \cdot m \cdot W\big)$ arithmetic operations, with n and m the number of vertices and moves, respectively, and W the maximal absolute weight in the game.

4.2 Optimal Strategy Improvement

The Optimal Strategy Improvement algorithm, presented in [Sch08a], optimizes the update mechanism of the strategy improvement approach (SI approach, for short) by performing more significantly improvement steps and, therefore, a lower total amount of steps. In the classic SI algorithm, the improvement step first evaluates the current strategy and then chooses a profitable modification. These two phases are distinct, this implies that the algorithm cannot predict the global effect of the individual modification and, in particular, the cross effect that may turn other modifications as non profitable. The optimal SI algorithm merges the two phases, adjusting the evaluation of the current strategy when a modification is applied. As a result, at every step, it selects the best combination of local modifications. This approach makes use of a pre-order of the evaluation of the positions that intuitively allows to apply a modification for a position after all of its successors have been already evaluated. The underlying data structure, that can be interpret as a sort of progress measure, is an estimation function e that associates a natural number to the game positions. The classic ordering relation $\leq$ and addition operation $+$ over the naturals are defined on the set of evaluations. The algorithm updates e until it satisfies the following conditions, where the operator $+$ adds the estimation value to the weight of an adjacent position:

1. $e(v) \leq e(u) + u$, for all adjacents $u \in Mv(v)$ of v, if $v \in \mathrm{Ps}_1$;
2. $e(v) \leq e(u) + u$, for some adjacent $u \in Mv(v)$ of v, if $v \in \mathrm{Ps}_0$;

Every improvement step computes the improvement p of the estimation e, such that the resulting estimation e' is equal to $e + p$. The step is performed according to the evaluation rules that ensure that the positions are evaluated in the optimal following order:

1. $v \in \mathrm{Ps}_1$ such that v has only evaluated successor, then compute the minimal improvement $p(v) = \mathsf{min}\{p(w) + (e(w) + w - e(v)) : w \in Mv(v)\}$;

2. $v \in \mathrm{Ps}_1$ such that v has an evaluated successor u whose improvement value $p(u)$ and the sum $(e(w) + w - e(v))$ are equal to 0, then also $p(v) = 0$;

3. $v \in \mathrm{Ps}_0$ such that v has only evaluated successor, then compute the maximal improvement $p(v) = \mathsf{max}\{p(w) + (e(w) + w - e(v)) : w \in Mv(v)\}$;

4. $v \in \mathrm{Ps}_1$ such that the improvement $p(v) = \mathsf{min}\{p(w) + (e(w) + w - e(v)) : w \in Mv(v)\}$ is minimal among evaluated successors.

When the procedure indentifies a position v, for which Player 0 has a winning strategy, the estimation of v assumes the top value ∞. This precisely happens when none of the above rules applies. When the improvement is equal to 0 for all the positions, the algorithm terminates. The positions v with $e(v) = \infty$ are won by Player 0, while the others for Player 1.

4.3 Small Energy Progress Measure

This approach, presented in [BCD+11], is based on the notion of Small Energy Progress Measures(SEPMs), that is very similar to the measure discussed in [Sch08a]. A SEPM is a bounded, integer, non negative function that impose local conditions between adjacent positions to ensure global properties in the graph. In the context of MPGs, such a property witnesses that Player 1 has a strategy to enforce that the mean payoff of every infinite path in the graph is non positive. Given a MPG, the SEPM μ assigns to each position $v \in \mathrm{Ps}$ a natural number of $\mathbb{N}$. The set of all possible values is extended with a maximum element ∞. The usual ordering relation $\leq$ and addition operation $+$ over the naturals are defined on this extended set. The aim of the algorithm is to assign a value to each position such that for all $v \in \mathrm{Ps}$ the following two conditions hold true:

1. $\mu(u) + v \leq \mu(v)$, for all adjacents $u \in Mv(v)$ of v, if $v \in \mathrm{Ps}_0$;

2. $\mu(u) + v \leq \mu(v)$, for some adjacent $u \in Mv(v)$ of v, if $v \in \mathrm{Ps}_1$.

Given a strategy $\sigma_\wp$ for player $\wp$, we denote with $\Game_{\sigma_\wp}$ the subgame of $\Game$ containing all and only the plays consistent with $\sigma_\wp$. This game is obtained from $\Game$ by removing all the moves $v' \in Mv(v)$ such that $v \in \mathrm{Ps}_\wp$ and $\sigma_\wp(v) \neq v'$.

Recalling that a *non-positive cycle* (*resp.*, positive cycle) in a game is a cycle with non positive total weight sum (*resp.*, strictly positive), we report the following two main results proved in [BCD$^+$11]. First, a strategy σ_1 for player 1 is winning if and only if there are no positive cycles in the subgame ∂_{σ_1}. Second, a game ∂ admits a SEPM if and only if there exist a strategy σ_1 such that the subgame ∂_{σ_1} has no positive cycles. As a consequence, a progress measure for player 1 exists if and only if player 1 has a winning strategy.

At the beginning of the computation, all the tuples are set to the minimum value, that is 0. Then, in order to obtain a SEPM, the value of every position is updated by means of a lifting function + that adds, when needed, the measure of an adjacent position to the weight of a given one. The intuition is that at the end of the process, the value of each position represents the sum of the weights encountered along the plays from that position. The lifting process described above is performed by the following operator:

$$\mathsf{lift}(\mu)(v) \triangleq \begin{cases} \mathsf{max}\{\mu(w) + v : w \in Mv(v)\}, & \text{if } v \in \mathrm{Ps_o}; \\ \mathsf{min}\{\mu(w) + v : w \in Mv(v)\}, & \text{otherwise.} \end{cases}$$

Assuming that a given position v has an adjacent with measure η, its lifting is performed according to the following rule: $\eta + v \triangleq \mathsf{max}\{0, \eta + \mathsf{wg}(v)\}$. Positions connected by positive cycles fall into a lifting loop that updates the value of the measures until the maximal value ∞ is reached. When the algorithm terminates, *i.e.* no lifting is possible, positions with defined values of μ are winning for player 1. While those with value ∞ are winning for player 0.

The authors of [BCD$^+$11] show that the algorithm requires $\mathrm{O}\big(n^2 \cdot W\big)$ lift operations in the worst case, with n the number of positions and W the maximal positive weight in the game, while the overall cost of these lift operations is $\mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W))$, with m the number of moves and $\mathrm{O}(\log n \cdot W)$ the cost of each arithmetic operation necessary to compute the update of the measures.

Currently, this is the tightest upper bound known on the time complexity of MPGs and a more detailed description of this algorithm is contained in Chapter 9.2.

Part II

Solving Games via Quasi Dominions

Chapter 5

Robust Worst Cases for Parity Games Algorithms

In this first chapter we present a new exponential-time worst-case family of games for the Recursive Algorithm that clearly shows the weaknesses of this classic approach, even if improved with memoization, scc-decomposition, and dominion decomposition of solved subgames. The discovery of such a robust worst case spurred us to the development of the new solution technique that is described in the next chapters.

This work has been presented at the

- 8th International Symposium on Games, Automata, Logics and Formal Verification (GANDALF'17), held in September 20-22, 2017, in Rome, Italy, with the title of "Robust Exponential Worst Cases for Divide-et-Impera Algorithms for Parity Games" and published on Volume 256 of EPTCS, pages 121-135, 2017.

The content of the chapter is based on the journal version of [BDM17] published in Information and Computation [BDM19a].

5.1 Abstract

The McNaughton-Zielonka *divide et impera* algorithm is the simplest and most flexible approach available in the literature for determining the winner in a parity game. Despite its theoretical exponential worst-case complexity and the negative reputation as a poorly effective algorithm in practice, it has been

shown to rank among the best techniques for the solution of such games. Also, it proved to be resistant to a lower bound attack, even more than the strategy improvements approaches, and only recently a family of games on which the algorithm requires exponential time has been provided by Friedmann. An easy analysis of this family shows that a simple memoization technique can help the algorithm solve the family in polynomial time. The same result can also be achieved by exploiting an approach based on the dominion-decomposition techniques proposed in the literature. These observations raise the question whether a suitable combination of dynamic programming and game-decomposition techniques can improve on the exponential worst case of the original algorithm. With this paper we answer the question negatively, by providing a robustly exponential worst case, showing that no possible intertwining of the above mentioned techniques can help mitigating the exponential nature of the *divide et impera* approaches. The resulting worst case is event more robust than that, since it serves as a lower bound for progress measures based algorithms as well, such as Small Progress Measure and its quasi-polynomial variant recently proposed by Jurdziński and Lazic.

5.2 The Worst-Case Core Family

Our goal is to define a family of parity games that exhibits worst-case behaviors for a number of parity games solvers and, at the same time, is robust with respect to memoization and game decomposition techniques possibly employed by the solvers. We start by focusing on satisfying the memoization resilience requirement, the most difficult one to meet. We define below a simple family that forces the Recursive algorithm to solve an exponential number of different subgames. As we shall see in the following two sections, this family is already robust enough to provide a worst-case for the Recursive algorithm, with or without memoization, and for progress measure-based algorithms such as Small Progress Measure [Jur00] and Succinct Progress Measure [JL17].

The *core family* $\{\mathfrak{D}^k_{\mathsf{C}}\}_{k=1}^{\omega}$ of games is informally defined as follows. For each $k \in \mathbb{N}_+$, game $\mathfrak{D}^k_{\mathsf{C}}$ contains $k+1$ gadgets, each one formed by three positions α_i, β_i, and γ_i, for $i \in [0,k]$. The positions β_i and γ_i, in gadget i, share the same priority i and opposite owners, namely player $i \bmod 2$ for β_i and $(i+1) \bmod 2$ for γ_i. The position α_i in the gadget has the same owner as the corresponding β_i. These positions are leading ones, having higher priorities than all the β's and γ's of the other gadgets. Positions within gadget i are connected as follows: α_i can only move to β_i; β_i can only move to γ_i; γ_i can choose either to move to β_i or to stay in γ_i itself. Two adjacent gadgets, of indexes i and $i+1$, are connected by only two moves: one from γ_i to α_{i+1} and one from β_{i+1} to α_i. Figure 5.1 depicts game $\mathfrak{D}^4_{\mathsf{C}}$. The gray portion in the figure represents the base game $\mathfrak{D}^1_{\mathsf{C}}$ of the family, where the priorities of all its α-positions have

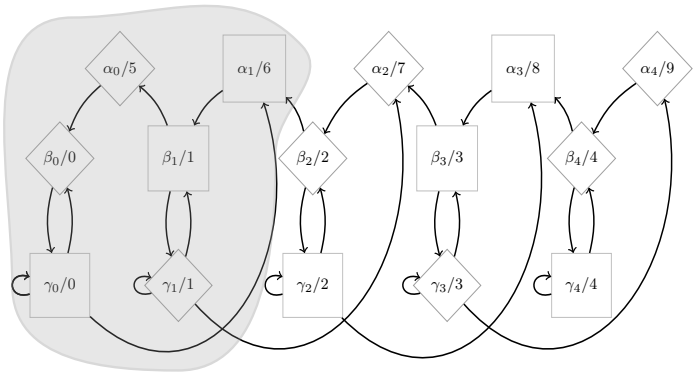


Figure 5.1: Game $\mathfrak{D}_{\mathsf{C}}^4$ of the core family.

been increased by 2, so as to comply with the requirement that the α-positions have the higher priorities. Indeed, in general, given an index k, game $\mathfrak{D}_{\mathsf{C}}^{k+1}$ is obtained from $\mathfrak{D}_{\mathsf{C}}^k$ by increasing by $2(k \bmod 2)$ units the priorities of each α_i in the gadgets of $\mathfrak{D}_{\mathsf{C}}^k$ and adding a new gadget with index $k+1$ connected as in figure. The games in the core family are formally described by the following definition.

Definition 5.2.1 (Core Family)**.** *The* core family $\{\mathfrak{D}_{\mathsf{C}}^k\}_{k=1}^{\omega}$*, where* $\mathfrak{D}_{\mathsf{C}}^k \triangleq \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle$*,* $\mathcal{A} \triangleq \langle \mathrm{Ps}^{\mathrm{o}}, \mathrm{Ps}^{1}, Mv \rangle$*, and* $\mathrm{Pr} \triangleq [0, 2k+1+k \bmod 2]$*, is defined as follows. For any index* $k \geq 1$*, the set of positions* $\mathrm{Ps} \triangleq \{\alpha_i, \beta_i, \gamma_i : 0 \leq i \leq k\}$ *of* $\mathfrak{D}_{\mathsf{C}}^k$ *is divided into three categories:*

- α_i *belongs to player* $\wp \triangleq i \bmod 2$*,* i.e.*,* $\alpha_i \in \mathrm{Ps}^{\wp}$*, and has priority* $\mathsf{pr}(\alpha_i) \triangleq k+i+1+k \bmod 2$*;*
- β_i *belongs to player* $\wp \triangleq i \bmod 2$*,* i.e.*,* $\beta_i \in \mathrm{Ps}^{\wp}$*, and has priority* $\mathsf{pr}(\beta_i) \triangleq i$*;*
- γ_i *belongs to player* $\overline{\wp} \triangleq (i+1) \bmod 2$*,* i.e.*,* $\gamma_i \in \mathrm{Ps}^{\overline{\wp}}$*, and has priority* $\mathsf{pr}(\gamma_i) \triangleq i$*.*

Moreover, the moves from positions α_i*,* β_i*, and* γ_i*, with* $0 \leq i \leq k$*, are prescribed as follows:*

- α_i *has a unique move to* β_i*,* i.e.*,* $Mv(\alpha_i) = \{\beta_i\}$*;*
- β_i *has one move to* γ_i *and one to* α_{i-1}*, if* $i > 0$*,* i.e.*,* $Mv(\beta_i) = \{\gamma_i\} \cup \{\alpha_{i-1} : i > 0\}$*;*
- γ_i *has one move to* γ_i *itself, one to* β_i*, and, if* $i < k$*, one to* α_{i+1}*,* i.e.*,* $Mv(\gamma_i) = \{\beta_i, \gamma_i\} \cup \{\alpha_{i+1} : i < k\}$*.*

It is not hard to verify that, for any $k \in \mathbb{N}_+$, the game $\mathfrak{D}^k_{\mathsf{C}}$ is completely won by player (k mod 2) and contains precisely $3(k+1)$ positions and $6k+4$ moves. As we shall see later in detail, the solution of each such games requires the Recursive algorithm to solve an exponential number of different subgames.

These core games form the backbone for a more general framework, consisting of an entire class of game families, with the property that each of them remains resilient to memoization techniques. Essentially, each game in any such family extends a core game. In order to define such a wider class, let us first establish what counts as a suitable extension of a core. Clearly, for a game $\mathfrak{D}$ to be an extension of $\mathfrak{D}^k_{\mathsf{C}}$, for some index $k \in \mathbb{N}_+$, it must contain $\mathfrak{D}^k_{\mathsf{C}}$ as a subgame. However, in order to prevent the Recursive algorithm from disrupting the core while processing $\mathfrak{D}$, we have to enforce some additional requirements. In particular, we need the algorithm to behave on $\mathfrak{D}$ virtually in the same way as it does on the core subgame. There is no unique way to ensure that. A simple solution is to require that all positions α_i still have the maximal priorities as in the core. Moreover, we require that no positions α_i and β_i have additional moves in $\mathfrak{D}$ *w.r.t.* those contained in the core. Finally, if γ_i can escape to some position v outside the core, then v does not have a higher priority, it has a move back to γ_i, and belongs to the opponent player *w.r.t.* γ_i. This condition ensures that no γ_i can decide to escape the core without being bounced back immediately by the opponent. The following definition makes the notion of extension precise.

Definition 5.2.2 (Core Extension)**.** *An arbitrary parity game $\mathfrak{D} \in \mathcal{P}$ is a* core extension *of $\mathfrak{D}^k_{\mathsf{C}}$, for a given index $k \in \mathbb{N}_+$, if the following four conditions hold:*

1. $\mathfrak{D} \setminus \mathrm{P} = \mathfrak{D}^k_{\mathsf{C}}$, *where* $\mathrm{P} \triangleq \{v \in \mathfrak{D} : v \notin \mathfrak{D}^k_{\mathsf{C}}\}$;
2. $\mathsf{pr}_{\mathfrak{D}}(v) < \mathsf{pr}_{\mathfrak{D}}(\alpha_0)$, *for all* $v \in \mathrm{P}$;
3. $\{\alpha_i, \beta_i \in \mathfrak{D} : 0 \leq i \leq k\} \cap (Mv(\mathrm{P}) \cup Mv^{-1}(\mathrm{P})) = \emptyset$;
4. $v \in \mathrm{Ps}^{\wp}$, $\gamma_i \in Mv(v)$, *and* $\mathsf{pr}(v) \leq i$, *for all* $v \in \mathrm{P} \cap Mv(\gamma_i)$, $i \in [0,k]$, *and* $\wp \triangleq i \bmod 2$.

We shall denote with $\mathcal{P}_{\mathsf{C}} \subseteq \mathcal{P}$ the set of all core extensions of $\mathfrak{D}^k_{\mathsf{C}}$, for any index $k \in \mathbb{N}_+$.

We can now define the abstract notion of worst-case family that extends the core family, while still preserving the same essential properties that we are going to prove shortly.

Definition 5.2.3 (Worst-Case Family)**.** *A family of parity games $\{\mathfrak{D}^k\}_{k=1}^{\omega}$ is a* worst-case family *if $\mathfrak{D}^k$ is a core extension of $\mathfrak{D}^k_{\mathsf{C}}$, for every index $k \in \mathbb{N}_+$.*

5.3 The Recursive Algorithm and the Core Family

In this section we discuss in more detail the algorithm of Zielonka previously introduced in Chapter 3.1. The Recursive procedure, reported in Algorithm 2 and proposed by Zielonka [Zie98] in an equivalent version, solves a parity game ∂ by decomposing it into two subgames, each of which is, then, solved recursively. Intuitively, the procedure works as follows. Algorithm 2, by means of Algorithm 3, starts by collecting all the positions that are forced to pass through a position with maximal priority $p \triangleq \mathsf{pr}(\partial)$ in that game. This first step results in computing the set $\mathrm{A} \triangleq \mathsf{atr}^{\wp}_{\partial}(\mathsf{pr}^{-1}_{\partial}(p))$, *i.e.*, the attractor to the set $\mathsf{pr}^{-1}_{\partial}(p)$ of positions with priority p *w.r.t.* player $\wp \triangleq p \bmod 2$. The subgame $\partial_{\mathtt{L}}$ is, then, obtained from ∂ by removing A from it and solved recursively. The result is a partitioning of the positions of $\partial_{\mathtt{L}}$ into two winning regions, $\mathrm{Wn}^{0}_{\mathtt{L}}$ and $\mathrm{Wn}^{1}_{\mathtt{L}}$, one per player.

Algorithm 2: Recursive algorithm.

signature $\mathsf{sol} : \mathcal{P} \to_{\partial} 2^{\mathrm{Ps}_{\partial}} \times 2^{\mathrm{Ps}_{\partial}}$

function $\mathsf{sol}(\partial)$

1 $(\partial_{\mathtt{L}}, \wp) \leftarrow \mathsf{f}_{\mathtt{L}}(\partial)$

2 $(\mathrm{Wn}^{0}_{\mathtt{L}}, \mathrm{Wn}^{1}_{\mathtt{L}}) \leftarrow \mathsf{sol}(\partial_{\mathtt{L}})$

3 **if** $\mathsf{pre}^{\overline{\wp}}_{\partial}(\mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}}) \setminus \mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}} = \emptyset$ **then**

4 $\quad (\mathrm{Wn}^{\wp}, \mathrm{Wn}^{\overline{\wp}}) \leftarrow (\mathrm{Ps}_{\partial} \setminus \mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}}, \mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}})$

else

5 $\quad \partial_{\mathtt{R}} \leftarrow \mathsf{f}_{\mathtt{R}}(\partial, \mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}}, \overline{\wp})$

6 $\quad (\mathrm{Wn}^{0}_{\mathtt{R}}, \mathrm{Wn}^{1}_{\mathtt{R}}) \leftarrow \mathsf{sol}(\partial_{\mathtt{R}})$

7 $\quad (\mathrm{Wn}^{\wp}, \mathrm{Wn}^{\overline{\wp}}) \leftarrow (\mathrm{Wn}^{\wp}_{\mathtt{R}}, \mathrm{Ps}_{\partial} \setminus \mathrm{Wn}^{\wp}_{\mathtt{R}})$

8 **return** $(\mathrm{Wn}^{0}, \mathrm{Wn}^{1})$

Algorithm 3: Left-subgame.

signature $\mathsf{f}_{\mathtt{L}} : \mathcal{P} \to \mathcal{P} \times \mathbb{B}$

function $\mathsf{f}_{\mathtt{L}}(\partial)$

1 $\wp \leftarrow \mathsf{pr}(\partial) \bmod 2$

2 $\partial^{\star} \leftarrow \partial \setminus \mathsf{atr}^{\wp}_{\partial}(\mathsf{pr}^{-1}_{\partial}(\mathsf{pr}(\partial)))$

3 **return** $(\partial^{\star}, \wp)$

Algorithm 4: Right-subgame.

signature $\mathsf{f}_{\mathtt{R}} : \mathcal{P} \times_{\partial} 2^{\mathrm{Ps}_{\partial}} \times \mathbb{B} \to \mathcal{P}$

function $\mathsf{f}_{\mathtt{R}}(\partial, \mathrm{W}, \wp)$

1 $\partial^{\star} \leftarrow \partial \setminus \mathsf{atr}^{\wp}_{\partial}(\mathrm{W})$

2 **return** $\partial^{\star}$

At this point, the algorithm checks whether the subgame $\partial_{\mathtt{L}}$ is completely won by $\wp$ or, more generally, if the adversary $\overline{\wp}$ cannot force any other position in ∂ into its own winning region $\mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}}$ in one move. In other words, none of the winning positions of the adversary $\overline{\wp}$ can attract something outside that region, *i.e.*, $\mathsf{pre}^{\overline{\wp}}_{\partial}(\mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}}) \setminus \mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}} = \emptyset$. If this is the case, the entire game ∂ is solved. Indeed, the positions of ∂ winning for $\wp$ are all its positions except, possibly, for those won by $\overline{\wp}$ in subgame $\partial_{\mathtt{L}}$ (see Line 4 of Algorithm 2). If, on the other hand, the above condition does not hold, the winning region of $\overline{\wp}$ can be extended with some other positions in ∂. Let $\mathrm{B} \triangleq \mathsf{atr}^{\overline{\wp}}_{\partial}(\mathrm{Wn}^{\overline{\wp}}_{\mathtt{L}})$ be the set collecting all such positions. Observe that all the positions in B are certainly winning for $\overline{\wp}$ in the entire game, as, from each such

position, $\overline{\wp}$ can force entering its own winning region $\mathrm{Wn}_{\mathtt{L}}^{\overline{\wp}}$, from which its opponent $\wp$ cannot escape. The residual subgame $\partial_{\mathtt{R}}$, obtained by removing B form ∂, as computed by Algorithm 4, may now contain positions winning for either player, and, therefore, needs to be solved again recursively (see Line 6 of Algorithm 2). All the positions of $\partial_{\mathtt{R}}$ that turn out to be winning for $\wp$ in that game, namely $\mathrm{Wn}_{\mathtt{R}}^{\wp}$, are, then, all and only those positions winning for $\wp$ in the entire game ∂, while the remaining ones are winning for $\overline{\wp}$ (see Line 7 of Algorithm 2).

As shown by Friedmann in [Fri11], the algorithm admits a worst case family of games $\{\partial^k\}_{k=1}^{\omega}$ that requires a number of recursive calls exponential in k. The reason is essentially the following. Each game ∂^k of that family contains all ∂^j, with $1 \leq j < k$, as subgames. Each recursive call that receives as input one such subgame ∂^j requires to eventually solve both ∂^{j-1} and ∂^{j-2}. As a consequence, the number of recursive calls performed by the algorithm on game ∂^k can be put in correspondence with a Fibonacci sequence. This proves that their number grows at least as fast as the sequence of the Fibonacci numbers, namely that their number is $\Omega\big(((1+\sqrt{5})/2)^k\big)$. The very reason that makes this family exponential also makes it amenable to a polynomial-time solution. It suffices to endow the Recursive algorithm with a memoization mechanism that, for each solved game ∂, records the triple $(\partial, \mathrm{Wn}_{\partial}^{0}, \mathrm{Wn}_{\partial}^{1})$. Each recursive call can, then, directly extract the winning regions of a subgame that is already contained in the collection, thus preventing the procedure from solving any subgame more than once. Not only does the resulting procedure make Friedman's worst case vain, but it also speeds up the solution of games significantly, as long as the number of repeated subgames remains relatively small, *e.g.*, linear in the size of the original game, which is often the case in practice.

We shall show that the same trick does not work for worst-case family defined in the previous section. In order to prove that any game in that family requires an exponential number of different subgames to be solved, we shall characterize a suitable subtree of the recursion tree generated by the algorithm, when called on one of the games in the family. Starting from the root, which contains the original game ∂^k, we fix specific observation points in the recursion tree that are identified by sequences in the set $w \in \{\mathtt{L}, \mathtt{R}\}^{\leq \lfloor k/2 \rfloor}$, where $\mathtt{L}$ (*resp.*, $\mathtt{R}$) denotes the recursive call on the left (*resp.*, right) subgame. Each sequence w identifies two subgames of ∂^k, $\widehat{\partial}_w^k$ and ∂_w^k, that correspond to the input subgames of two successive nested calls. In the analysis of the recursion tree, we only take into account the left subgame ∂_w^k of each $\widehat{\partial}_w^k$, thus disregarding its right subtree as it is inessential to the argument. An example of the resulting subgame tree for game $\partial_{\mathtt{C}}^2$ of the core family is depicted in Figure 5.2. According to Algorithm 2 on input $\partial_{\mathtt{C}}^2 = \partial_{\varepsilon}^2$, the first (left) recursive call is executed on the subgame obtained by removing the 1-attractor to the positions with maximal priority, in this case α_2, which only contains α_2. Therefore, the left subgame coincides precisely with $\widehat{\partial}_{\mathtt{L}}^2$. The second call is executed on the game

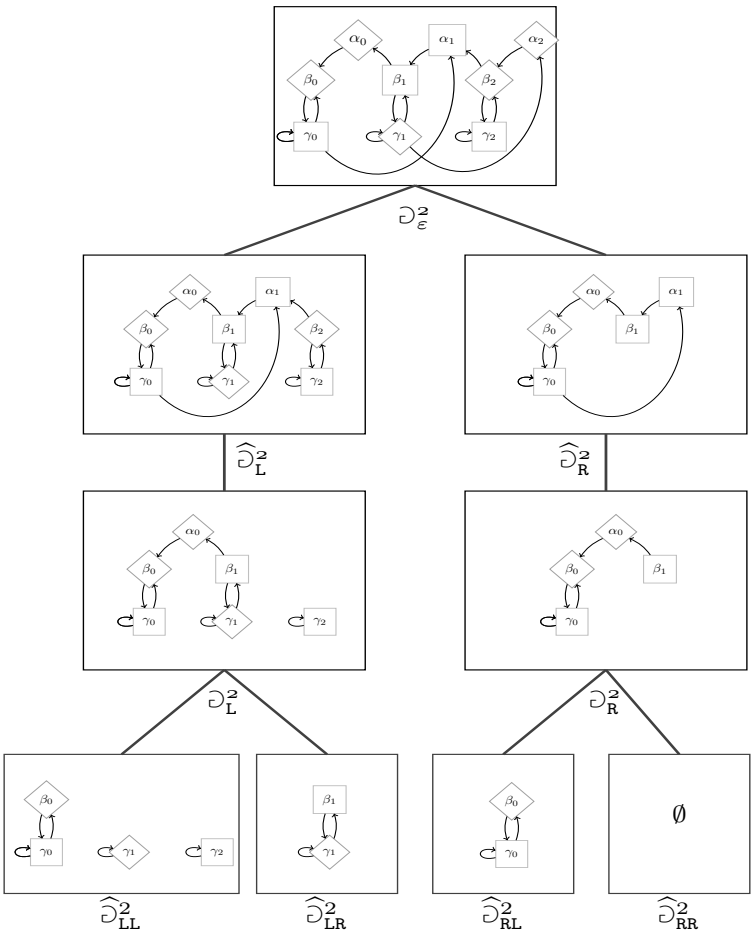


Figure 5.2: The induced subgame tree G^2 of ∂^2.

obtained by removing the 0-attractor in ∂^2_ε to the winning region for player 0 of the left subgame $\widehat{\partial}^2_{\mathtt{L}}$. In this case, that winning region is precisely $\{\beta_2, \gamma_2\}$, and its 0-attractor is $\{\alpha_2, \gamma_1, \beta_2, \gamma_2\}$. As consequence, the subgame fed to the right-hand call precisely coincides with $\widehat{\partial}^2_{\mathtt{R}}$. The rest of the subtree is generated applying the same reasoning. The following definition generalizes this notion to a game of any worst-case family and characterizes the portion of the recursion tree we are interested in analyzing.

Definition 5.3.1 (Induced Subgame Tree)**.** *Given a worst-case family* $\{\partial^k\}_{k=1}^{\omega}$*, the* induced subgame tree $\mathsf{G}^k \triangleq \{\partial^k_w\}_{w\in\{\mathtt{L},\mathtt{R}\}^{\leq k}} \cup \{\widehat{\partial}^k_w\}^{w\neq\varepsilon}_{w\in\{\mathtt{L},\mathtt{R}\}^{\leq k+1}}$ w.r.t. *an index* $k \in \mathbb{N}_+$ *is defined inductively on the structure of the sequence* $w \in \{\mathtt{L},\mathtt{R}\}^{\leq \lfloor k/2 \rfloor}$ *as follows, where* $\partial^k_\varepsilon \triangleq \partial^k$ *and* $z \triangleq k - 2|w|$*:*

1. $\widehat{\mathfrak{I}}^k_{w\mathtt{L}} \triangleq \mathfrak{I}^k_w \setminus \mathsf{atr}^{\overline{\wp}_k}_{\mathfrak{I}^k_w}(\{\alpha_z\})$;

2. $\widehat{\mathfrak{I}}^k_{w\mathtt{R}} \triangleq \mathfrak{I}^k_w \setminus \mathsf{atr}^{\wp_k}_{\mathfrak{I}^k_w}(\mathrm{Wn}^{\wp_k}_{\widehat{\mathfrak{I}}^k_{w\mathtt{L}}})$;

3. $\mathfrak{I}^k_w \triangleq \widehat{\mathfrak{I}}^k_w \setminus \mathsf{atr}^{\wp_k}_{\widehat{\mathfrak{I}}^k_w}(\{\alpha_{z+1}\})$, *if* $w \neq \epsilon$.

Before proceeding with the proof of the main result of this section, we need some additional properties of the induced subgame tree of any worst-case family. The following lemma, which is essential to the result, states some invariants of the elements contained in the tree induced by a game $\mathfrak{I}^k$ that extends the core $\mathfrak{I}^k_{\mathtt{C}}$. In particular, these invariants ensure that all those elements are subgames of $\mathfrak{I}^k$ (Items 1 and 4) and that, depending on the identifying sequence w, they contain the required leading positions α_i of the core (Items 2 and 7). In addition, it states two important properties of every left child in the tree, *i.e.*, those elements identified by a sequence w ending with $\mathtt{L}$. Both of them will be instrumental in proving that all the subgames in the tree are indeed different and to assess their number, as we shall see in Lemmas 5.3.3 and 5.3.4. The first property ensures that each such game necessarily contains a specific position γ_i, with index i depending on w (Items 3 and 5). The second one (Item 6) characterizes the winning region for player $\wp_k$ of the left-child subgames $\widehat{\mathfrak{I}}^k_{w\mathtt{L}}$. It states that, in each such game, the winning positions for player $\wp_k$ contained in the corresponding core $\mathfrak{I}^k_{\mathtt{C}}$ are all its β-positions and γ-positions, with index of the same parity as $\wp_k$ and greater than the maximal index x of the leading α-position. Indeed, as soon as the higher positions α_i, with $i \in [x+1, k]$, are removed from the game, each residual corresponding $\gamma_{\wp_k+2j}$, possibly together with its associated $\beta_{\wp_k+2j}$, is necessarily contained in an independent $\wp_k$-dominion.

In the sequel, by $\mathsf{lst}(w)$ we denoted the last position of a non-empty sequence $w \in \{\mathtt{L}, \mathtt{R}\}^+$.

Lemma 5.3.1. *For any index* $k \in \mathbb{N}_+$ *and sequence* $w \in \{\mathtt{L}, \mathtt{R}\}^{\leq \lfloor k/2 \rfloor + 1}$, *let* $z \triangleq k - 2|w|$. *Then, the following properties hold:*

- *if* $|w| \leq \lfloor k/2 \rfloor$, *then*

  1. $\mathfrak{I}^k_w$ *is a subgame of* $\mathfrak{I}^k$;

  2. $\alpha_j \in \mathfrak{I}^k_w \iff j \in [0, z]$;

  3. $\gamma_j \in \mathfrak{I}^k_w$, *for all* $j \in [0, z]$, *and* $\gamma_{z+1} \in \mathfrak{I}^k_w$, *if* $w \neq \varepsilon$ *and* $\mathsf{lst}(w) = \mathtt{L}$;

- *if* $|w| > 0$, *then*

  4. $\widehat{\mathfrak{I}}^k_w$ *is a subgame of* $\mathfrak{I}^k$;

5. $\gamma_j \in \widehat{\mathfrak{I}}^k_w$, *for all* $j \in [0, z]$, *and, whenever* $\mathsf{lst}(w) = \mathtt{L}$, $\gamma_{z+1} \in \widehat{\mathfrak{I}}^k_w$, *if* $|w| \leq \lfloor k/2 \rfloor$, *and* $\gamma_0 \in \widehat{\mathfrak{I}}^k_w$, *otherwise;*

6. $\mathrm{Wn}^{\wp_k}_{\widehat{\mathfrak{I}}^k_w} \cap \mathfrak{I}^k_{\mathtt{C}} = \{\beta_{z+2j}, \gamma_{z+2j} \in \widehat{\mathfrak{I}}^k_w : j \in [1, |w|]\}$, *if* $\mathsf{lst}(w) = \mathtt{L}$;

- *if* $0 < |w| \leq \lfloor k/2 \rfloor$, *then*

7. $\alpha_j \in \widehat{\mathfrak{I}}^k_w \iff j \in [0, z+1]$.

Proof. The proof proceeds by induction on the structure of the sequence w. For the base case $w = \varepsilon$, we have that $\mathfrak{I}^k_w = \mathfrak{I}^k$ and $z = k$. Thus, Items 1, 2, and 3 hold due to Item 1 of Definition 5.2.2 and the structure of the game core $\mathfrak{I}^k_{\mathtt{C}}$ given in Definition 5.2.1, while Items 4,5, 6, and 7 are vacuously verified. For the inductive case, assume that all properties hold for a given sequence $w \in \{\mathtt{L}, \mathtt{R}\}^{\leq \lfloor k/2 \rfloor}$. We show that they hold for $\overline{w} = wx \in \{w\mathtt{L}, w\mathtt{R}\}$ as well. By Items 1 and 2 of Definition 5.3.1, we have that $\widehat{\mathfrak{I}}^k_{\overline{w}} = \mathfrak{I}^k_w \setminus \mathrm{A}$, where $\mathrm{A} = \mathsf{atr}^{\overline{\wp_k}}_{\mathfrak{I}^k_w}(\{\alpha_z\})$, if $x = \mathtt{L}$, and $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\mathfrak{I}^k_w}(\mathrm{Wn}^{\wp_k}_{\widehat{\mathfrak{I}}^k_{w\mathtt{L}}})$, otherwise. So, Item 4 for $\overline{w}$ immediately follows from Item 1 for w. Moreover, Item 1, together with Items 1 and 3 of Definition 5.2.2 and the topology of the game $\mathfrak{I}^k_{\mathtt{C}}$, implies that $\mathrm{A} = \{\alpha_z, \beta_{z+1}\}$, if $x = \mathtt{L}$, and $\mathrm{A} \cap \mathfrak{I}^k_{\mathtt{C}} = \{\alpha_z\} \cup \{\gamma_{z-1} : |w| < \lfloor k/2 \rfloor\} \cup \{\beta_{z+1} : \gamma_{z+1} \notin \mathfrak{I}^k_w\} \cup \{\beta_{z+2j}, \gamma_{z+2j} \in \mathfrak{I}^k_w : j \in [0, |w|]\}$, otherwise. Indeed, if $x = \mathtt{L}$, the position β_{z+1} is the only one that gets attracted by α_z *w.r.t.* player $\overline{\wp_k}$, as the other two positions γ_{z-1} and γ_{z+1} having a non-forced move to either α_z or β_{z+1} belong to player $\wp_k$. If, on the other hand, $x = \mathtt{R}$, due to Item 6 for $w\mathtt{L}$, we have that $\mathrm{Wn}^{\wp_k}_{\widehat{\mathfrak{I}}^k_{w\mathtt{L}}} \cap \mathfrak{I}^k_{\mathtt{C}} = \{\beta_{z+2j}, \gamma_{z+2j} \in \widehat{\mathfrak{I}}^k_{w\mathtt{L}} : j \in [0, |w|]\}$. Therefore, the only positions of $\mathfrak{I}^k_w \cap \mathfrak{I}^k_{\mathtt{C}}$ attracted by $\mathrm{Wn}^{\wp_k}_{\widehat{\mathfrak{I}}^k_{w\mathtt{L}}}$ *w.r.t.* player $\wp_k$ are α_z and, possibly, γ_{z-1} or β_{z+1}. Indeed, α_z has a forced move to β_z, while γ_{z-1}, if $|w| < k$, is willing to follow α_z, and, finally, if γ_{z+1} does not belong to the game, β_{z+1} is forced to reach α_z as well. At this point, Item 7 for $\overline{w}$ is immediately derived from Item 2 for w, by observing that in both cases α_z is the only α-position that is removed and $k - 2|\overline{w}| + 1 = k - 2|w| - 1 < z$. Similarly, Item 5 for $\overline{w}$ is derived from Item 3 for w, by observing that the only γ-positions possibly removed have index at least equal to $z - 1 > k - 2|\overline{w}| = z - 2$. We can now focus on the games $\mathfrak{I}^k_{\overline{w}}$ that, by Definition 5.3.1, are equal to $\widehat{\mathfrak{I}}^k_{\overline{w}} \setminus \mathrm{A}$ with $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\widehat{\mathfrak{I}}^k_{\overline{w}}}(\{\alpha_{z-1}\})$. Obviously, Item 1 for $\overline{w}$ simply derives from Item 4 for $\overline{w}$. Again by Items 1 and 3 of Definition 5.2.2 and the topology of the game $\mathfrak{I}^k_{\mathtt{C}}$, we have that $\mathrm{A} = \{\alpha_{k-2|\overline{w}|+1}, \beta_{k-2|\overline{w}|}\} = \{\alpha_{z-1}, \beta_z\}$, if $x = \mathtt{L}$, and $\mathrm{A} = \{\alpha_{k-2|\overline{w}|+1}\} = \{\alpha_{z-1}\}$, otherwise. Consequently, Item 2 for $\overline{w}$ follows from Item 7 for $\overline{w}$, since α_{z-1} is the unique α-position that is removed from the game. Similarly, Item 3 for $\overline{w}$ is implied by Item 5 for $\overline{w}$, as no γ-positions is removed.

To conclude the proof, we need to show Item 5 for $\overline{w} = w\mathtt{L}$. First observe that, due to Items 1 and 4 of Definition 5.2.2, the set of positions $\mathrm{D} = \{\beta_{z+2j}, \gamma_{z+2j} \in \widehat{\mathfrak{I}}^k_{\overline{w}} : j \in [0, |w|]\} \cup \bigcup^{j \in [0,|w|]}_{\gamma_{z+2j} \in \widehat{\mathfrak{I}}^k_{\overline{w}}} Mv(\gamma_{z+2j}) \cap \widehat{\mathfrak{I}}^k_{\overline{w}}$

is a $\wp_k$-dominion in the game $\widehat{\partial}^k_{\overline{w}}$, constituted by the possibly overlapping union of the $\wp_k$-dominions $\mathrm{D}^{\wp_k}_j = \{\beta_{z+2j}, \gamma_{z+2j} \in \widehat{\partial}^k_{\overline{w}}\} \cup Mv(\gamma_{z+2j}) \cap \widehat{\partial}^k_{\overline{w}}$, for every index $j \in [0, |w|]$ such that $\gamma_{z+2j} \in \widehat{\partial}^k_{\overline{w}}$. Indeed, the priority of the positions in $\{\beta_{z+2j}, \gamma_{z+2j} \in \widehat{\partial}^k_{\overline{w}}\}$ is of the same parity of player $\wp_k$, being equal to $z + 2j$, and the priorities of the positions in $Mv(\gamma_{z+2j}) \cap \widehat{\partial}^k_{\overline{w}}$ is not greater than $z + 2j$. Moreover, there is no move the opponent $\overline{\wp_k}$ can take from its unique position γ_{z+2j} in order to escape from the set, as there is no position $\alpha_{z+2j+1} \in \widehat{\partial}^k_{\overline{w}}$. Hence, $\mathrm{D}^{\wp_k}_j$ is, by definition, a $\wp_k$-dominion.

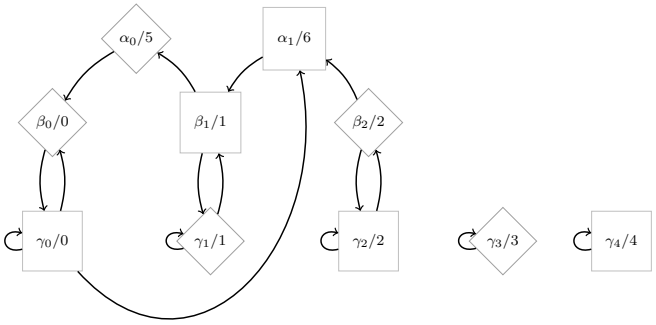


Figure 5.3: Game $\widehat{\partial}^4_{\mathtt{LL}}$.

For example, looking at the game $\partial^4_{\mathtt{LL}}$ depicted in Figure 5.3, it is immediate to see that β_2, γ_2, and γ_4 belong to the winning region of player 0. Observe that $\mathrm{D} \cap \partial^k_{\mathtt{C}} = \{\beta_{k-2|\overline{w}|+2j}, \gamma_{k-2|\overline{w}|+2j} \in \widehat{\partial}^k_{\overline{w}} : j \in [1, |\overline{w}|]\}$, being $k - 2|\overline{w}| = z - 2$. Therefore, to prove the thesis, it sufficies to show that all the other positions in $\partial^k_{\mathtt{C}} \setminus \mathrm{D}$ belong to the $\overline{\wp_k}$-dominion in the game $\widehat{\partial}^k_{\overline{w}}$ obtained by the possibly overlapping union of the $\overline{\wp_k}$-dominions $\mathrm{D}^{\overline{\wp_k}}_j = \{\beta_{z+2j+1}, \gamma_{z+2j+1} \in \widehat{\partial}^k_{\overline{w}}\} \cup Mv(\gamma_{z+2j+1}) \cap \widehat{\partial}^k_{\overline{w}}$, for every index $j \in [0, |w|[$ such that $\gamma_{z+2j+1} \in \widehat{\partial}^k_{\overline{w}}$, together with the $\overline{\wp_k}$-dominion $\mathrm{H} = \{\alpha_j, \beta_j, \gamma_j : j \in [0, z[\} \cup \bigcup^{j \not\equiv_2 z}_{j \in [0, z[} Mv(\gamma_j) \cap \widehat{\partial}^k_{\overline{w}}$, where $\bigcup^{j \not\equiv_2 z}_{j \in [0, z[} Mv(\gamma_j) \cap \widehat{\partial}^k_{\overline{w}}$ takes care of including those position of the core extension that are not in the core but are attracted by the γ-positions in $\{\alpha_j, \beta_j, \gamma_j : j \in [0, z[\}$. For instance, in the game of Figure 5.3, where $\overline{w} = \mathtt{LL}$ and $\overline{\wp_k} = \overline{\wp_4} = 1$, such a 1-dominion is the union of $\mathrm{D}^1_1 = \{\gamma_3\}$ and $\mathrm{H} = \{\alpha_j, \beta_j, \gamma_j : j \in \{0, 1\}\}$. The proof that $\mathrm{D}^{\overline{\wp_k}}_j$ is a $\overline{\wp_k}$-dominion is, *mutatis mutandis*, the same of the one used to show that $\mathrm{D}^{\wp_k}_j$ is a $\wp_k$-dominion. Therefore, we only focus on the component H. Observe that, since $\alpha_z \notin \widehat{\partial}^k_{\overline{w}}$, the two positions $\beta_{z-1}, \gamma_{z-1} \in \mathrm{H}$, together with those in $Mv(\gamma_{z-1}) \cap \widehat{\partial}^k_{\overline{w}}$, form a $\overline{\wp_k}$-dominion X. Again, the proof here is the same of that used for the $\overline{\wp_k}$-dominions $\mathrm{D}^{\overline{\wp_k}}_j$. Consequently, the $\overline{\wp_k}$-attractor $\mathrm{A} = \mathsf{atr}^{\overline{\wp_k}}_{\widehat{\partial}^k_{\overline{w}}}(\mathrm{X})$ is still a $\overline{\wp_k}$-dominion. Now, by structural induction, we can also see that the remaining positions in $\mathrm{H} \setminus \mathrm{A}$ correspond to another the $\overline{\wp_k}$-dominion. Therefore, in $\widehat{\partial}^k_{\overline{w}}$ player $\wp_k$ has only the possibility to either remain in $\mathrm{H} \setminus \mathrm{A}$ or to pass in A and then remain there forever. Hence, H is a $\overline{\wp_k}$-dominion as required. □

Finally, the next lemma simply establishes that all the subgames contained in the induced subgame tree of Definition 5.3.1 are indeed generated by the Recursive algorithm when called with input game $\mathcal{D}^k$ of some worst-case family.

Lemma 5.3.2. *For any index $k \in \mathbb{N}_+$ and sequence $w \in \{\mathtt{L},\mathtt{R}\}^{\leq \lfloor k/2 \rfloor}$, the following properties hold:*

1. $\mathsf{f}_{\mathtt{L}}(\mathcal{D}^k_w) = (\widehat{\mathcal{D}}^k_{w\mathtt{L}}, \overline{\wp_k})$;
2. $\mathsf{f}_{\mathtt{R}}(\mathcal{D}^k_w, \mathrm{Wn}^{\wp_k}_{\widehat{\mathcal{D}}^k_{w\mathtt{L}}}, \wp_k) = \widehat{\mathcal{D}}^k_{w\mathtt{R}}$;
3. $\mathsf{f}_{\mathtt{L}}(\widehat{\mathcal{D}}^k_w) = (\mathcal{D}^k_w, \wp_k)$, *if* $w \neq \varepsilon$.

Proof. The proof of each item easily follows from the definition of the algorithm and the properties of the game they are applied to.

[Item 1]. Due to Items 1 and 2 of Lemma 5.3.1 and Items 1 and 2 of Definition 5.2.2, the maximal priority $\mathsf{pr}(\mathcal{D}^k_w)$ of the game $\mathcal{D}^k_w$ is $2(k-|w|)+1+\wp_k \equiv_2 \overline{\wp_k}$, which is assumed by the position $\alpha_{k-2|w|}$. Consequently, we have $\mathsf{pr}^{-1}(\mathsf{pr}(\mathcal{D}^k_w)) = \{\alpha_{k-2|w|}\}$. Thus, the thesis follows due to Item 1 of Definition 5.3.1 and the instructions constituting Algorithm 3.

[Item 2]. The thesis is an immediate consequence of the structure of Algorithm 4 and Item 2 of Definition 5.3.1.

[Item 3]. Due to Items 4 and 7 of Lemma 5.3.1 and Items 1 and 2 of Definition 5.2.2, the maximal priority $\mathsf{pr}(\widehat{\mathcal{D}}^k_w)$ of the game $\widehat{\mathcal{D}}^k_w$ is $2(k-|w|)+2+\wp_k \equiv_2 \wp_k$, which is assumed by the position $\alpha_{k-2|w|+1}$. Consequently, we have $\mathsf{pr}^{-1}(\mathsf{pr}(\widehat{\mathcal{D}}^k_w)) = \{\alpha_{k-2|w|+1}\}$. Thus, the thesis follows due to Item 3 of Definition 5.3.1 and the instructions constituting Algorithm 3. □

We are now ready for the main result of this section, namely that the induced subgame tree contains elements which are all different from each other and whose number is exponential in the index k. We split the result into two lemmas. The first one simply states that any subgame in the left subtree of some $\mathcal{D}^k_w$ is different from any other subgame in the right subtree. The idea is that for for any subgame in the tree, all subgames of its left subtree contain at least one position, a specific position γ_i, with i depending on w, that is not contained in any subgame of its right subtree.

Lemma 5.3.3. *For all indexes $k \in \mathbb{N}_+$ and sequences $w, v \in \{\mathtt{L},\mathtt{R}\}^*$, with $\ell \triangleq |w|+|v| \leq \lfloor k/2 \rfloor$ and $z \triangleq k-2|w|-1$, the following properties hold:*

1. *$\gamma_z \in \mathcal{D}^k_{w\mathtt{L}v}$ and $\gamma_z \in \widehat{\mathcal{D}}^k_{w\mathtt{L}v}$, if $\ell < \lfloor k/2 \rfloor$, and $\gamma_0 \in \widehat{\mathcal{D}}^k_{w\mathtt{L}v}$, otherwise;*
2. *$\gamma_z \notin \mathcal{D}^k_{w\mathtt{R}v}$ and $\gamma_z \notin \widehat{\mathcal{D}}^k_{w\mathtt{R}v}$, if $\ell < \lfloor k/2 \rfloor$, and $\gamma_0 \notin \widehat{\mathcal{D}}^k_{w\mathtt{R}v}$, otherwise.*

Proof. First observe that, if $\ell < \lfloor k/2 \rfloor$, by Items 3 and 5 of Lemma 5.3.1, position γ_z belongs to both $\eth^k_w$ and $\widehat{\eth}^k_w$, since $0 < z < k - 2|w|$. Let us consider Item 1 of the current lemma first and show that the every position γ_z belongs to all the descendants of $\eth^k_w$ in its left subtree. The proof proceeds by induction on the length of the sequence v and recall that, due to Item 2 (*resp.*, 7) of Lemma 5.3.1, the position with maximal priority in $\eth^k_w$ (*resp.*, in $\widehat{\eth}^k_w$) is α_z (*resp.*, α_{z+1}). Assume, for the base case, that $|v| = 0$. The thesis becomes $\gamma_z \in \eth^k_{w\mathtt{L}}$ and $\gamma_z \in \widehat{\eth}^k_{w\mathtt{L}}$. By Item 1 of Definition 5.3.1, $\widehat{\eth}^k_{w\mathtt{L}}$ is defined as $\eth^k_w \setminus \mathrm{A}$, where $\mathrm{A} = \mathsf{atr}^{\overline{\wp_k}}_{\eth^k_w}(\{\alpha_{z+1}\})$. By Definition 5.2.2 of core extension (Items 1, 3, and 4), a move entering α_{z+1} may only come from β_{z+2}, if it is present in the subgame, which is always the case unless $w = \varepsilon$, or from γ_z, whose owner is player $\wp_k$ and cannot be attracted. Hence, $\mathrm{A} = \{\alpha_{z+1}, \beta_{z+2}\}$, if $w \neq \varepsilon$, and $\mathrm{A} = \{\alpha_{z+1}\}$, otherwise. Similarly, by Item 3, $\eth^k_{w\mathtt{L}}$ is defined as $\widehat{\eth}^k_{w\mathtt{L}} \setminus \mathrm{A}$, where $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\widehat{\eth}^k_{w\mathtt{L}}}(\{\alpha_z\})$. For the same observations as in the previous case, we have that $\mathrm{A} = \{\alpha_z, \beta_{z+1}\}$. Hence, no position γ_i is removed from either game and the thesis immediately follows.

For the inductive case, assume $|v| > 0$, let $v \triangleq v' \cdot x$, with $x \in \{\mathtt{L}, \mathtt{R}\}$, and $\overline{w} \triangleq w\mathtt{L}v'$. By the inductive hypothesis, $\gamma_z \in \widehat{\eth}^k_{\overline{w}}$ and $\gamma_z \in \eth^k_{\overline{w}}$. We have two cases, depending on whether $x = \mathtt{L}$ or $x = \mathtt{R}$. Let $r \triangleq k - 2|\overline{w}|$. If $x = \mathtt{L}$, according to Item 1 of Definition 5.3.1, $\widehat{\eth}^k_{\overline{w}\mathtt{L}}$ is obtained from $\eth^k_{\overline{w}}$ by removing $\mathrm{A} = \mathsf{atr}^{\overline{\wp_k}}_{\eth^k_{\overline{w}}}(\{\alpha_r\}) = \{\alpha_r, \beta_{r+1}\}$. Similarly, by Item 3 of Definition 5.3.1, $\eth^k_{\overline{w}\mathtt{L}}$ is defined as $\widehat{\eth}^k_{\overline{w}\mathtt{L}} \setminus \mathrm{A}$, where $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{L}}}(\{\alpha_{r-1}\})$, by observing that $|\overline{w}\mathtt{L}| = |\overline{w}| + 1$ and, therefore, $k - 2|\overline{w}\mathtt{L}| + 1 = r - 1$. In both cases the thesis follows immediately.

Let us now consider the case with $x = \mathtt{R}$. According to Item 2 of Definition 5.3.1, $\widehat{\eth}^k_{\overline{w}\mathtt{R}}$ is obtained by removing the set $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\eth^k_{\overline{w}}}(\mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{L}}})$ from $\eth^k_{\overline{w}}$. Position γ_z has index z congruent to $\overline{\wp_k}$ modulo 2 and cannot belong to $\mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{L}}}$, which, by Item 6 of Lemma 5.3.1, only contains, among the positions from the core, those β_i and γ_i, with $i \geq k - 2|w| > z$ and congruent to $\wp_k$ modulo 2. Since, $\widehat{\eth}^k_{\overline{w}\mathtt{L}}$ is a subgame of a core extension, it holds that γ_z is owned by player $\wp_k$ and can only have a move leading to α_{z+1}, which is not in the subgame, or to a position outside the core and owned by player $\overline{\wp_k}$. As a consequence, it cannot end up in $\mathsf{atr}^{\wp_k}_{\eth^k_{\overline{w}}}(\mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{L}}})$ and the thesis holds.

Finally, recall that $\eth^k_{\overline{w}\mathtt{R}} = \widehat{\eth}^k_{\overline{w}\mathtt{R}} \setminus \mathrm{A}$, where $\mathrm{A} = \mathsf{atr}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{R}}}(\{\alpha_{\hat{r}}\})$, with $\hat{r} \triangleq k - 2|\overline{w}\mathtt{R}| + 1$. In the considered subgame, $\alpha_{\hat{r}}$ has incoming moves only from $\beta_{\hat{r}+1}$ and $\gamma_{\hat{r}-1}$. However, $\hat{r} - 1 = k - 2|\overline{w}\mathtt{R}| < k - 2|w| - 1 = z$. Moreover, index $\hat{r}+1$ is congruent to $\wp_k$ modulo 2, and thus $\beta_{\hat{r}+1}$ is not contained in $\widehat{\eth}^k_{\overline{w}\mathtt{R}}$, being in $\mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{\overline{w}\mathtt{L}}}$ as shown above. As a consequence, $\mathrm{A} = \{\alpha_{\hat{r}}\}$ and the thesis follows. In addition, when $|w| = \lfloor k/2 \rfloor$, Item 3 of Lemma 5.3.1 tells us that $\gamma_0 \in \eth^k_w$. If $\ell = k$, instead, we have that γ_0 belongs to $\widehat{\eth}^k_{w\mathtt{L}}$, due to Item 5 of Lemma 5.3.1. This ends the proof of Item 1 of the lemma. As to Item 2 of the lemma, first observe that, if $|w| < \lfloor k/2 \rfloor$, then $\gamma_z \notin \widehat{\eth}^k_{w\mathtt{R}}$. Indeed, position $\gamma_z \in \mathsf{atr}^{\wp_k}_{\eth^k_w}(\mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{w\mathtt{L}}})$, as shown above. Since

this set is removed from $\eth^k_w$ to obtain $\widehat{\eth}^k_{w\mathtt{R}}$, the thesis holds for $\widehat{\eth}^k_{w\mathtt{R}}$. Moreover, every descendant of $\widehat{\eth}^k_{w\mathtt{R}}$ in the subgame tree is obtained only by removing positions. As a consequence, none of them can contain position γ_z. In case $|w| = \lfloor k/2 \rfloor$, instead, it suffices to observe that, according to Item 6 of Lemma 5.3.1, $\gamma_0 \in \mathrm{Wn}^{\wp_k}_{\widehat{\eth}^k_{w\mathtt{L}}}$, hence it cannot be contained in $\widehat{\eth}^k_{w\mathtt{R}}$. □

The result asserting the exponential size of the induced subtrees of any worst-case family is given by the next lemma. This follows by observing that the number of nodes in the induced tree is exponential in k and by showing that the subgames associated with any two nodes in the tree G^k are indeed different.

Lemma 5.3.4. *$|\mathsf{G}^k| = 3(2^{\lfloor k/2 \rfloor + 1} - 1)$, for any $k \in \mathbb{N}_+$.*

Proof. To prove that the size of G^k is as stated, we first need to show that all the elements contained in the subgame trees are different, namely that, for each $w \neq w'$, the subgames $\eth^k_w$, $\eth^k_{w'}$, $\widehat{\eth}^k_w$, and $\widehat{\eth}^k_{w'}$ are pairwise different. Let us start by showing that $\eth^k_w \neq \widehat{\eth}^k_w$, for each $w \neq \varepsilon$. By Item 7 of Lemma 5.3.1, position $\alpha_{k-2|w|+1} \in \widehat{\eth}^k_w$ and, by Item 3 of Definition 5.3.1, this position is removed from $\widehat{\eth}^k_w$ to obtain $\eth^k_w$. Hence, those two subgames cannot be equal. Let us consider now two subgames, each associated with one of the sequences w and w'. There are two possible cases: either *(i)* w is a strict prefix of w', *i.e.*, one subgame is a descendant of the other in the subgame tree, or *(ii)* w and w' share a common longest prefix $\overline{w}$ that is different from both, *i.e.*, the two subgames lie in two distinct subtrees of the subgame associated with $\overline{w}$. In case *(i)* we have that $w' = wv$, for some $v \neq \varepsilon$. An easy induction on the length of v can prove that if $\eth \in \{\eth^k_w, \widehat{\eth}^k_w\}$ and each $\eth' \in \{\eth^k_{w'}, \widehat{\eth}^k_{w'}\}$ are the subgames associated with w and w', respectively, then the second is a strict subgame of the first, *i.e.*, $\eth' \subset \eth$. Indeed, Definition 5.3.1 together with Items 2, 7, and 6 of Lemma 5.3.1 ensure that, at each step downward along a path in the tree starting from $\eth$, whether we proceed on the left or the right branch, at least one position is always removed from the current subgame. In case *(ii)*, instead, Item 1 of Lemma 5.3.3 tells us that there is at least one position, γ_z in the lemma, contained in all the subgames of the left subtree, while Item 2 states that the same position is not contained in any subgame of the right subtree. Therefore, each of the two subgames associated with w must be different from either of the two subgames associated with w'.

Finally, to prove the statement of the lemma, it suffices to observe that the number of sequences of length at most $\lfloor k/2 \rfloor$ over the alphabet $\{\mathtt{L}, \mathtt{R}\}$ are precisely $2^{\lfloor k/2 \rfloor + 1} - 1$ and with each such sequence w a subgame $\eth^k_w$ is associated. As a consequence, the set $\{\eth^k_w\}_{w \in \{\mathtt{L},\mathtt{R}\}^{\leq k}}$ contains $2^{\lfloor k/2 \rfloor + 1} - 1$ different elements. Moreover, each such subgame has two children in $\{\widehat{\eth}^k_w\}^{w \neq \varepsilon}_{w \in \{\mathtt{L},\mathtt{R}\}^{\leq k+1}}$. We can, then, conclude that the size of G^k is precisely $3(2^{\lfloor k/2 \rfloor + 1} - 1)$. □

As a consequence of Lemma 5.3.4, we can obtain a stronger lower bound for the execution time of

the Recursive algorithm. Indeed, the result holds regardless of whether the algorithm is coupled with a memoization technique.

Theorem 5.3.1 (Exponential Worst Case)**.** *The number of distinct recursive calls executed by the Recursive algorithm, with or without memoization, on a game with n positions is* $\Omega\big(2^{\frac{n}{6}}\big)$ *in the worst case.*

Proof. To prove the theorem, it suffices to consider a game $\mathfrak{I}^k_{\mathsf{C}}$ belonging to the core family. Indeed, Lemma 5.3.2 states that the induced subgame tree of $\mathfrak{I}^k_{\mathsf{C}}$ is a subset of the recursion tree induced by the Recursive algorithm executed on that game. Therefore, according to Lemma 5.3.4, the algorithm performs at least $3(2^{\lfloor k/2 \rfloor+1}-1)$ calls, each on a different subgame. By Definition 5.2.1, game $\mathfrak{I}^k_{\mathsf{C}}$ has $n = 3(k+1)$ positions and, therefore, we have $\lfloor k/2 \rfloor = \lfloor \frac{n-3}{6} \rfloor$. As a consequence, the number of recursive calls is bounded from below by $3(2^{\lfloor \frac{n+3}{6} \rfloor}-1) = \Omega\big(2^{\frac{n}{6}}\big)$. □

5.4 Progress Measure-based Algorithms and the Core Family

Progress measures are decorations of a graph, whose local consistency usually guarantees, from an high-level point of view, some global property of the graph itself [KK91]. Fixed a player $\wp \in \{0,1\}$, the original progress measure algorithm [Jur00] and its quasi-polynomial time variation [JL17] exploit this idea, by decorating every position $v \in \mathrm{Ps}$ of a game $\mathfrak{I}$ with an element $\iota(v) \in \mathrm{M}^{\wp}$, called *measure*, from an ordered set $(\mathrm{M}^{\wp}, \sqsubseteq)$. These decorations ι, called *$\wp$-measure functions*, naturally inherit the order $\sqsubseteq$ from $\mathrm{M}^{\wp}$. Measures are used to compare, via a suitable truncated ordering $\sqsubseteq_{\mathsf{pr}(v)}$, the relevance of the priority $\mathsf{pr}(v)$ of position $v \in \mathrm{Ps}$ *w.r.t.* those associated with its successors $v' \in Mv(v)$. These comparisons characterize, in this context, the local consistency property among positions mentioned above. It can be proved that a minimal measure function ι_ω satisfying such a property always exists and uniquely identifies those positions in Ps belonging to the winning region $\mathrm{Wn}^{\wp}$ of player $\wp$ [Jur00]. In more detail, the two algorithms share the same structure of the ordered measure set $(\mathrm{M}^{\wp}, \sqsubseteq)$, where *(i)* the support is $\mathrm{M}^{\wp} \subseteq \{\top\} \cup \mathrm{D} \to \mathrm{N}$, *(ii)* $\mathrm{D} \triangleq \{d \in \mathrm{Pr} : d \equiv_2 \wp\}$ is the set of priorities congruent to the parity of player $\wp$, *(iii)* $(\mathrm{N}, \preceq)$ is an ordered set, *(iv)* $\sqsubseteq$ is the lexicographic ordering *w.r.t.* $\preceq$ on $\mathrm{M}^{\wp} \setminus \{\top\}$ that also satisfies $m \sqsubseteq \top$, for every $m \in \mathrm{M}^{\wp}$, and *(v)* $\top \in \mathrm{M}^{\wp}$. The priority-sensitive comparison $\sqsubseteq_p$, with $p \in \mathrm{Pr}$, is derived from $\sqsubseteq$ and a truncation operation defined as follows: $m{\downarrow_p} \triangleq m{\restriction}\{d \in \mathrm{D} : d \geq p\}$, if $m \neq \top$, and $m{\downarrow_p} \triangleq \top$, otherwise, for all $m \in \mathrm{M}^{\wp}$. The truncated ordering $\sqsubseteq_p$ can, then, be obtained as $m_1 \sqsubseteq_p m_2$ if $m_1{\downarrow_p} \sqsubseteq m_2{\downarrow_p}$, for all $m_1, m_2 \in \mathrm{M}^{\wp}$. Given a $\wp$-measure function $\iota \colon \mathrm{Ps} \to \mathrm{M}^{\wp}$, *i.e.*, a function that assigns a measure in $\mathrm{M}^{\wp}$ to each position of the game, the notion of *successor measure*

of ι *w.r.t.* a move $(v_1, v_2) \in Mv$ and an ordering relation $\lhd \in \{\sqsubset, \sqsubseteq\}$ can be defined as the element $\downarrow_{\lhd}(\iota)((v_1, v_2)) \triangleq \mathsf{min}_{\sqsubseteq_{\mathsf{pr}(v_1)}}\{m \in \mathrm{M}^{\wp} : \iota(v_2) \lhd m\}$. Notice that, if $\iota(v_2) = \top$, then $\downarrow_{\lhd}(\iota)((v_1, v_2)) = \top$, since $\{m \in \mathrm{M}^{\wp} : \iota(v_2) \lhd m\} = \emptyset$ in this case. Thanks to the Knaster-Tarski theorem, the minimal measure function ι_ω satisfying the local consistency property can be obtained as the least fixed point $\iota_\omega = \mu X.\mathsf{lift}^{\wp}(X)$ of the monotone *lifting operator* $\mathsf{lift}^{\wp}\colon (\mathrm{Ps} \to \mathrm{M}^{\wp}) \to (\mathrm{Ps} \to \mathrm{M}^{\wp})$ defined via the two auxiliary functions $\mathsf{lift}_{\lhd}\colon (\mathrm{Ps} \to \mathrm{M}^{\wp}) \to (Mv \to \mathrm{M}^{\wp})$ and $\mathsf{lift}^{\wp}\colon (\mathrm{Ps} \to \mathrm{M}^{\wp}) \to (Mv \to \mathrm{M}^{\wp})$ according to the following:

Definition 5.4.1.

1. $\mathsf{lift}_{\lhd}(\iota)((v_1, v_2)) \triangleq \mathsf{max}_{\sqsubseteq_{\mathsf{pr}(v_1)}}\{\iota(v_1), \downarrow_{\lhd}(\iota)((v_1, v_2))\}$;
2. $\mathsf{lift}^{\wp}(\iota)((v_1, v_2)) \triangleq \begin{cases} \mathsf{lift}_{\sqsubset_{\mathsf{pr}(v_1)}}(\iota)((v_1, v_2)), & \text{if } \mathsf{pr}(v_1) \equiv_2 \wp; \\ \mathsf{lift}_{\sqsubseteq_{\mathsf{pr}(v_1)}}(\iota)((v_1, v_2)), & \text{otherwise}; \end{cases}$
3. $\mathsf{lift}^{\wp}(\iota)(v) \triangleq \begin{cases} \mathsf{max}_{\sqsubseteq_{\mathsf{pr}(v_1)}}\{\mathsf{lift}^{\wp}(\iota)((v, v')) : (v, v') \in Mv\}, & \text{if } v \in \mathrm{Ps}_{\wp}; \\ \mathsf{min}_{\sqsubseteq_{\mathsf{pr}(v_1)}}\{\mathsf{lift}^{\wp}(\iota)((v, v')) : (v, v') \in Mv\}, & \text{otherwise.} \end{cases}$

In their works, the authors of [Jur00] and [JL17] proved that the set of winning positions $\mathrm{Wn}^{\wp}$ of player $\wp$ in a parity game coincides with the set $\{v \in \mathrm{Ps} : \iota_\omega(v) = \top\}$ of positions with measure $\top$ in the fixpoint measure function, while the remaining ones $\mathrm{Wn}^{\overline{\wp}} = \mathrm{Ps} \setminus \mathrm{Wn}^{\wp}$ are winning for player $\overline{\wp}$.

Thanks to the monotonicity of the successor function, and independently of the definition of the ordered set $(\mathrm{N}, \preceq)$, we can prove that, in a game belonging to any extension of the Core family, the lift of a $\wp$-measure function ι always increments by the least possible quantity the measure associated with position $\gamma_{\wp}$, as stated by the following lemma.

Lemma 5.4.1. *Let $\supset^k$ be an element in a worst case family $\{\supset^k\}_{k\geq 1}^{\omega}$ and $\wp \in \{0, 1\}$ a player. Then, $\mathsf{lift}^{\wp}(\iota)(\gamma_{\wp}) \in \{\iota(\gamma_{\wp}), \downarrow_{\sqsubset}(\iota)((\gamma_{\wp}, \gamma_{\wp}))\}$, for any measure function $\iota \in \mathrm{M}^{\wp}$.*

Proof. By Definition 5.2.1, the position $\gamma_{\wp}$ has priority $\mathsf{pr}(\gamma_{\wp}) = \wp$, but player $\overline{\wp}$. Therefore, due to Definition 5.4.1, we have that $\mathsf{lift}^{\wp}(\iota)(\gamma_{\wp}) = \mathsf{min}_{\sqsubseteq_{\mathsf{pr}(\gamma_{\wp})}}\{\mathsf{lift}^{\wp}(\iota)((\gamma_{\wp}, v)) : (\gamma_{\wp}, v) \in Mv\}$. Now, if there is successor v of $\gamma_{\wp}$ with $\iota(v) \sqsubset_{\wp} \iota(\gamma_{\wp})$, it holds that $\mathsf{lift}_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v)) = \mathsf{max}_{\sqsubseteq_{\mathsf{pr}(\gamma_{\wp})}}\{\iota(\gamma_{\wp}), \downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v))\} = \iota(\gamma_{\wp})$, since $\iota(v) \sqsubset_{\wp} \iota(\gamma_{\wp})$ implies $\downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v)) \sqsubseteq_{\mathsf{pr}(\gamma_{\wp})} \iota(\gamma_{\wp})$. Otherwise, $\iota(\gamma_{\wp}) \sqsubseteq_{\mathsf{pr}(\gamma_{\wp})} \iota(v)$, for every possible successor v. Hence, $\mathsf{lift}_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v)) = \mathsf{max}_{\sqsubseteq_{\mathsf{pr}(\gamma_{\wp})}}\{\iota(\gamma_{\wp}), \downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v))\} = \downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, \gamma_{\wp}))$. Observe that, in the last equality we are exploiting the monotonicity of the successor function, *i.e.*, $\downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v_1)) \sqsubseteq_{\mathsf{pr}(\gamma_{\wp})} \downarrow_{\sqsubset_{\wp}}(\iota)((\gamma_{\wp}, v_2))$, if $\iota(v_1) \sqsubseteq_{\mathsf{pr}(\gamma_{\wp})} \iota(v_2)$. At this point, since $\wp \in \{0, 1\}$, it holds

that $\gamma_\wp$ has the minimal priority of parity $\wp$ in the game, thus, $\sqsubset_{\mathsf{pr}(\gamma_\wp)}{=}\sqsubset$. As a consequence, we have that $\mathsf{lift}_{\sqsubset_\wp}(\iota)((\gamma_\wp, v)) = \downarrow_\sqsubset(\iota)((\gamma_\wp, \gamma_\wp))$. □

As mentioned above, the Small Progress Measure algorithm and its succinct version only differ in the definitions of the underlying ordered set $(\mathrm{N}, \preceq)$ and of the measure set $\mathrm{M}^\wp$. For Small Progress Measure, we have that $(\mathrm{N}, \preceq) \triangleq (\mathbb{N}, \leq)$ and, for all $m \in \mathrm{M}^\wp \setminus \{\top\}$ and $d \in \mathrm{D}$, it holds that $m(d) \leq n_d$, where n_d denotes the number of positions with priority d in the game. For the Succinct Progress Measure algorithm, instead, we have that $\mathrm{N} \triangleq \{0,1\}^*$, $\preceq$ is the lexicographic ordering on bit sequences *w.r.t.* the ordering $0 < \epsilon < 1$, and, for all $m \in \mathrm{M}^\wp \setminus \{\top\}$, it holds that $\sum_{d\in\mathrm{D}} |m(d)| \leq \lceil \log_2 n \rceil$, where n is the total number of positions in the game.

We can prove that any worst case family $\{\Game^k\}^\omega_{k\geq 1}$ contains difficult instance for the both these algorithms as well. In both cases, this is proved by exploiting Lemma 5.4.1 and by showing that position $\gamma_\wp$ is lifted an exponential (*resp.*, a quasi-polynomial) number of times before a fixpoint is reached.

Lemma 5.4.2 (Small Progress Measure Exponential Worst Case)**.** *The number of lifts executed by the Small Progress Measure algorithm for player $\wp \in \{0,1\}$ on the $(2k+\wp)$-th element of a worst case family is $\Omega\big(6^k\big)$.*

Proof. Given a worst-case family $\{\Game^k\}^\omega_{k\geq 1}$, let us fix a player $\wp$ and the subfamily $\{\Game^{2k+\wp}\}^\omega_{k\geq 1}$. Moreover, observe that, each game in the subfamily is completely won by player $\wp$. The Small Progress Measure algorithm on a game $\Game^r$ with $r = 2k + \wp$ for player $\wp$ computes the least fixpoint $\iota_\omega = \mu\iota.\mathsf{lift}^\wp(\iota)$ of $\mathsf{lift}^\wp$, which assigns each position v of the game to $\top$, *i.e.*, $\iota_\omega(v) = \top$. To do this, the algorithm iteratively applies the lift operator, *i.e.*, $\iota_{i+1} = \mathsf{lift}^\wp(\iota_i)$, starting from the initial measure function ι_0, which assigns the smallest measure in $\mathrm{M}^\wp$. By Lemma 5.4.1, we know that $\mathsf{lift}^\wp(\iota)(\gamma_\wp) \in \{\iota(\gamma_\wp), \downarrow_\sqsubset(\iota)((\gamma_\wp, \gamma_\wp))\}$, for every ι. In addition, the priority of $\gamma_\wp$ is either 0 or 1, depending on $\wp$. As a consequence, $\sqsubseteq_{\mathsf{pr}(\gamma_\wp)}{=}\sqsubseteq$, in other words, no truncation occurs. Therefore, the sequence $\iota_0, \iota_1, \ldots, \iota_h = \iota_\omega$ of lifts performed by the algorithm induces a non-decreasing chain $m_0 = \iota_0(\gamma_\wp), m_1 = \iota_1(\gamma_\wp), \ldots, m_h = \iota_h(\gamma_\wp)$ of measures for $\gamma_\wp$, which contains a maximal strictly increasing subchain $m_{j_0}, m_{j_1}, \ldots, m_{j_z}$, where *(i)* $m_{j_0} = m_0$, *(ii)* $m_{j_{l+1}}$ is the minimal measure greater than m_{j_l}, *i.e.*, $m_{j_{l+1}} = \mathsf{min}_\sqsubseteq\{m \in \mathrm{M}^\wp : m_{j_l} \sqsubset m\}$, for $l \in [0, z]$, and *(iii)* $m_{j_z} = \top$. Therefore, z is equal to the height of the total ordered set $\mathrm{M}^\wp$, which corresponds to its cardinality. For the game $\Game^r$, a measure $m \in \mathrm{M}^\wp \setminus \{\top\}$ is isomorphic to a $(r+1)$-tuple $\langle b_{k+\wp-1}, \ldots, b_0, t_k, \ldots, t_0\rangle$ of numbers, with $b_i \in \{0,1\}$ and $t_i \in \mathrm{T} \supseteq \{0,1,2\}$ [Jur00]. Indeed, there is a single position for each one of the $k+\wp$ priorities of parity $\wp$ in the range $[r+\wp+1, 2r+\wp+1]$, while there are at least two positions for the $k+1$ priorities of the same parity in $[0, r]$. The cardinality of $\mathrm{M}^\wp$ is, therefore, at least $2^{(k+\wp)}3^{(k+1)} \geq 6^k$. □

Theorem 5.4.1 (Small Progress Measure Exponential Worst Case)**.** *The number of lifts executed by the Small Progress Measure algorithm on a game with n positions is $\Omega\big(6^{\frac{n}{6}}\big)$ in the worst case.*

Proof. Consider the core family $\{\mathfrak{I}_{\mathsf{C}}^{k}\}_{k\geq 1}^{\omega}$ and its $\mathfrak{I}_{\mathsf{C}}^{2k+\wp}$ element containing $n = 3(2k+\wp+1)$ positions. By Lemma 5.4.2, we have that Small Progress Measure performs $\Omega\big(6^{k}\big)$ lifts, where $k = \frac{n-3\wp-3}{6}$. Consequently, the number of lifts is $\Omega\big(6^{\frac{n}{6}}\big)$. □

Lemma 5.4.3 (Succinct Progress Measure Quasi-Polynomial Worst Case)**.** *The number of lifts executed by the Succinct Progress Measure algorithm for player $\wp \in \{0,1\}$ on the $(2k+\wp)$-th element of a worst case family is $\Omega\big((6k)^{\log k - \log\lceil \log 6k\rceil}\big)$.*

Proof. Due to the previously described framework, it is clear that the Succinct Progress Measure algorithm is structurally identical to the Small Progress Measure one, being the measure set the only difference between the two approaches. As a consequence, the number of lifts required by the former algorithm on an element $\mathfrak{I}^{r}$ with index $r = 2k+\wp$ of a worst-case family $\{\mathfrak{I}^{k}\}_{k\geq 1}^{\omega}$ is equal to the dimension of its measure set $\mathrm{M}^{\wp}$ as well. Now, every measure $m \in \mathrm{M}^{\wp} \setminus \{\top\}$ is isomorphic to a $(r+1)$-tuple $\langle s_0, \ldots, s_r\rangle$ of binary strings [JL17], with $s_i \in \{0,1\}^*$ and $\sum_{i=0}^{r} |s_i| = \lceil \log n\rceil$, where $n \geq n^\star \triangleq 3(r+1)$ is the number of positions in $\mathfrak{I}^{r}$. For any $h \in [0, \lceil \log n\rceil]$, there are 2^h binary strings $s \in \{0,1\}^h$ of length h and, for each one of them, exactly $\binom{h+r}{h}$ ways to be spit into $(r+1)$ substrings $s_0, \ldots, s_r$. Consequently, the size of the measure set $\mathrm{M}^{\wp}$ is $\sum_{h=0}^{\lceil \log n\rceil} 2^h \binom{h+r}{h} \geq n^\star \binom{\lceil \log n^\star\rceil + r}{\lceil \log n^\star\rceil} \geq n^\star \left(\frac{\lceil \log n^\star\rceil + r}{\lceil \log n^\star\rceil}\right)^{\lceil \log n^\star\rceil} \geq n^\star \left(\frac{r}{\lceil \log n^\star\rceil}\right)^{\lceil \log n^\star\rceil} \geq n^\star \left(\frac{r}{\lceil \log n^\star\rceil}\right)^{\log n^\star}$. Since $r \geq \frac{n^\star}{6}$, for any $k \in \mathbb{N}_+$, we have

$$
\begin{aligned}
|\mathrm{M}^{\wp}| &\geq n^\star \left(\frac{n^\star}{6\lceil \log n^\star\rceil}\right)^{\log n^\star} = n^\star \left(\frac{n^\star}{n^{\star \log_{n^\star}(6\lceil \log n^\star\rceil)}}\right)^{\log n^\star} \\
&= n^{\star (\log n^\star)(1-\log_{n^\star}(6\lceil \log n^\star\rceil))+1} = n^{\star \log n^\star - \log(6\lceil \log n^\star\rceil))+1} \\
&= n^{\star \log n^\star - \log\lceil \log n^\star\rceil - \log 6 + 1}.
\end{aligned}
$$

Finally, since $n^\star > 6k$, we have that

$$
\begin{aligned}
|\mathrm{M}^{\wp}| &\geq (6k)^{\log 6k - \log\lceil \log 6k\rceil - \log 6 + 1} = (6k)^{\log k - \log\lceil \log 6k\rceil + 1} \\
&= \Omega\Big((6k)^{\log k - \log\lceil \log 6k\rceil}\Big).
\end{aligned}
$$

□

Theorem 5.4.2 (Succinct Progress Measure Quasi-Polynomial Worst Case)**.** *The number of lifts executed by the Succinct Progress Measure algorithm, on a game with n positions is $\Omega\big((n-6)^{\log \frac{n}{6} - \log\lceil \log n\rceil}\big)$ in the worst case.*

Proof. Consider the core family $\{\mathfrak{D}^k_{\mathtt{C}}\}^{\omega}_{k\geq 1}$ and its $\mathfrak{D}^{2k+\wp}_{\mathtt{C}}$ element containing $n = 3(2k + \wp + 1)$ positions. By Lemma 5.4.3, we have that Succinct Progress Measure performs $\Omega\big((6k)^{\log k - \log\lceil \log 6k\rceil}\big)$ lifts, where $k = \frac{n-3\wp-3}{6}$. Consequently, the number of lifts is $\Omega\big((n-6)^{\log \frac{n}{6} - \log\lceil \log n\rceil}\big)$. □

The quasi-polynomial time algorithm proposed in [FJS+17] is yet another instance of a progress measure based approach and can easily be recast in the above framework, by a suitable definition of the measure set $\mathrm{M}^{\wp}$, the truncation operation $m{\downarrow_p}$, and the successor function $\downarrow_{\lhd}$. Unlike the two algorithm discussed above, however, the resulting successor function does not enjoy the monotonicity property, as pointed out in [FJS+17]. As a consequence, Lemma 5.4.1 does not hold for this approach. Nonetheless, we conjecture that the number of lifts for position $\gamma_{\wp}$ is still quasi-polynomial in the index of the game and an analogous result of Theorem 5.4.2 holds in this case as well.

5.5 SCC-Decomposition Resilient Games

The previous sections provide a class of parametric families of parity games over which a dynamic-programming approach cannot help improving the asymptotic exponential behavior of the classic Recursive algorithm and that serves a an exponential, for SPM, or a quasi-polynomial, for Succinct SPM, lower bound for the progress measures based algorithms. The core family, however, is not robust enough to resist to game decomposition techniques such as, *e.g.*, SCC-decomposition. In particular, it is not hard to observe that for the Recursive algorithm a SCC-decomposition of the underlying game graph, if applied by each recursive call as described in [FL09], would disrupt the recursive structure of the core family $\{\mathfrak{D}^k_{\mathtt{C}}\}^{\omega}_{k=1}$ and, consequently, break the exponential worst-case. This is due to the fact that the subgames $\mathfrak{D}^k_w$ in the induced subgame tree get decomposed into distinct SCCs, which can then be solved as independent subgames and memoized. In other words, the Recursive algorithm extended with memoization and SCC-decomposition can easily solve the core family. A concrete instance of this behavior can be observed by looking at the two games $\widehat{\mathfrak{D}}^1_{\mathtt{LL}}$ and $\widehat{\mathfrak{D}}^1_{\mathtt{RL}}$ of Figure 5.2. The first one is formed by three distinct components, one of which exactly corresponds to $\widehat{\mathfrak{D}}^1_{\mathtt{RL}}$. Therefore, a solution of these components immediately implies that the leaves in the induced subgame tree could not be considered distinct games *w.r.t.* to the behavior of the combined algorithm anymore. This behavior can, however, be prevented by introducing a suitable extension of the core family, which complies with the requirements of Definition 5.2.2. The basic idea is to connect all the pairs of positions γ_i and γ_j together in a clique-like fashion, by means of additional positions, denoted $\delta^{\wp}_{\{i,j\}}$ with $\wp \in \{0, 1\}$, whose owners $\wp$ are chosen so as to preserve the exponential behavior on the underlying core family. With more detail, if $i \equiv_2 j$, there

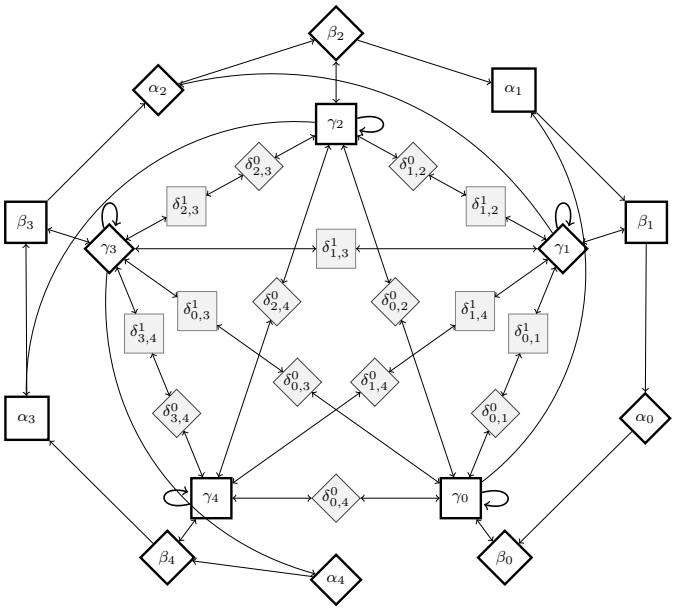


Figure 5.4: Game ∂^4_{S} of the SCC family.

is a unique connecting position $\delta^{\wp}_{\{i,j\}}$ of parity $\wp \equiv_2 i$, the opposite of that of γ_i and γ_j. If, on the other hand, $i \not\equiv_2 j$, two mutually connected positions, $\{\delta^0_{\{i,j\}}, \delta^1_{\{i,j\}}\}$, separate γ_i and γ_j. Figure 5.4 depicts the extension ∂^4_{S} of the core game ∂^4_{C}. The grey-filled positions in the figure enforce mutual connection among the nodes of the core in any subgame of the subgame tree, while the remaining positions are exactly those of the Core family and connected among them in the same way. The complete formalization of the new family follows.

Definition 5.5.1 (SCC Family)**.** *The* SCC family $\{\partial^k_{\mathsf{S}}\}_{k=1}^{\omega}$*, where* $\partial^k_{\mathsf{S}} \triangleq \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle$*,* $\mathcal{A} \triangleq \langle \mathrm{Ps}^{\mathrm{o}}, \mathrm{Ps}^{1}, Mv \rangle$*,* $\mathrm{Pr} \triangleq [0, 2k+1+\wp]$*, and* $\mathrm{Ps} \triangleq \partial^k_{\mathsf{C}} \cup \mathrm{P}$*, is defined, for any index* $k \geq 1$*, as follows:*

1. $\mathrm{P} \triangleq \{\delta^{\wp}_{\{i,j\}} : \wp \in \{0,1\} \wedge i,j \in [0,k] \wedge i \neq j \wedge (i \equiv_2 j \rightarrow \wp \equiv_2 i)\}$*;*
2. $(\gamma_i, \delta^{\wp}_{\mathrm{I}}), (\delta^{\wp}_{\mathrm{I}}, \gamma_i) \in Mv \iff i \in \mathrm{I}$ *and* $i \equiv_2 \wp$*, for* $i \in [0,k]$ *and* $\delta^{\wp}_{\mathrm{I}} \in \mathrm{P}$*;*
3. $(\delta^{\wp}_{\mathrm{I}}, \delta^{\overline{\wp}}_{\mathrm{I}}) \in Mv \iff i \not\equiv_2 j$*, for* $\delta^{\wp}_{\mathrm{I}} \in \mathrm{P}$ *and* $\mathrm{I} = \{i,j\}$*;*
4. $\delta^{\wp}_{\mathrm{I}} \in \mathrm{Ps}^{\wp}$ *and* $\mathsf{pr}(\delta^{\wp}_{\mathrm{I}}) \triangleq 0$*, for* $\delta^{\wp}_{\mathrm{I}} \in \mathrm{P}$*;*
5. $\partial^k_{\mathsf{S}} \setminus \mathrm{P} = \partial^k_{\mathsf{C}}$.

Intuitively, in Item 1, P denotes the set of additional positions of $\mathfrak{D}^k_{\mathsf{S}}$ *w.r.t.* to the core family game $\mathfrak{D}^k_{\mathsf{C}}$, which is, indeed, a proper subgame, as stated in Item 5. Item 2, instead, formalizes the moves connecting the additional $\delta^{\wp}_{\mathrm{I}}$ positions with the γ_i of the core, while Item 3 describes the mutual connection between the δ positions that share the same doubleton of indexes I. Finally, Item 4 associates each $\delta^{\wp}_{\mathrm{I}}$ with its corresponding owner $\wp$ and priority 0. The following lemma proves that such a parity-game family is indeed a worst-case family.

Lemma 5.5.1. *The SCC family* $\{\mathfrak{D}^k_{\mathsf{S}}\}_{k=1}^{\omega}$ *is a worst-case family.*

Proof. To prove that the *SCC family* of Definition 5.5.1 is a worst-case family, we need to show that each game $\mathfrak{D}^k_{\mathsf{S}}$ is a core extension of $\mathfrak{D}^k_{\mathsf{C}}$, *i.e.*, that it complies with the Definition 5.2.2. Items 1 and 5 of Definition 5.5.1 imply Item 1 of Definition 5.2.2, since the set P does not contain any position of the core and, in addition, $\mathfrak{D}^k_{\mathsf{C}}$ is a subgame of $\mathfrak{D}^k_{\mathsf{S}}$, as all the positions and moves of the core are contained in $\mathfrak{D}^k_{\mathsf{S}}$. Item 2 of Definition 5.2.2 follows from Item 4 of Definition 5.5.1. Indeed, by Definition 5.2.1, $\mathsf{pr}(\alpha_i) \triangleq k + i + \wp + 1$, for $i \in [0, k]$ and $k \geq 1$, while all the additional positions in P have priority 0. By Items 2 and 3 of Definition 5.5.1, there are no moves connecting positions δ_{I} with positions α_i or β_i, for any $i \in [0, k]$, hence Item 3 of Definition 5.2.2 is satisfied. Finally, we need to show that whenever a γ_i has a move to a position v in P, then v does not have higher priority, belongs to the opponent of γ_i, and has a move back to γ_i (Item 4 of Definition 5.2.2). This property is enforced by Items 2 and 4 of Definition 5.5.1. Indeed, by the latter, position $\delta^{\wp}_{\mathrm{I}}$ is owned by player $\wp$. Moreover, by the former, γ_i, whose owner is $(i + 1) \bmod 2$, can only have a move to $\delta^{\wp}_{\{i,j\}}$ if $\delta^{\wp}_{\{i,j\}}$ has a move back to γ_i and $\wp = i \bmod 2$. Hence, the two positions belong to opposite players. Since, in addition, all positions δ have priority 0, the requirement is satisfied. □

Due to the clique-like structure of the new family, it is not hard to see that every game in the induced subgame tree forms a single SCC. This guarantees that the intertwining of SCC-decomposition and memoization cannot prevent an exponential worst-case behavior of the Recursive algorithm on this family.

Lemma 5.5.2. *Each game in the induced subgame tree* G^k *of the SCC family* $\{\mathfrak{D}^k_{\mathsf{S}}\}_{k=1}^{\omega}$*, for an arbitrary index* $k \in \mathbb{N}_+$*, forms a single SCC.*

Proof. Let $\mathfrak{D}^k_w \in \mathsf{G}^k$ (*resp.*, $\widehat{\mathfrak{D}}^k_w \in \mathsf{G}^k$) be a game in the induced subgame tree. By induction on the structure of the string w, it is not hard to see that, for all indexes $i, j \in [0, k]$ with $i \neq j$, it holds that $\gamma_i, \gamma_j \in \mathfrak{D}^k_w$ (*resp.*, $\gamma_i, \gamma_j \in \widehat{\mathfrak{D}}^k_w$) iff the positions $\delta^{\wp}_{\{i,j\}} \in \mathrm{P}$, with $\wp \in \{0, 1\}$, belong to $\mathfrak{D}^k_w$ (*resp.*, $\widehat{\mathfrak{D}}^k_w$), as well. For the base case $\mathfrak{D}^k_{\varepsilon} = \mathfrak{D}^k$, the thesis trivially follows from Definition 5.5.1. For the inductive

case $\widehat{\partial}^k_{wx} = \partial^k_w \setminus \mathrm{A}$ (*resp.*, $\partial^k_{wx} = \widehat{\partial}^k_w \setminus \mathrm{A}$), let us assume, as inductive hypothesis, that the statement holds for ∂^k_w (*resp.*, $\widehat{\partial}^k_w$). By Definition 5.3.1, the set A is computed as the attractor to some set of positions B such that either *(i)* $\gamma_i, \gamma_j, \delta^{\wp}_{\{i,j\}} \notin \mathrm{A}$ or *(ii)* $\delta^{\wp}_{\{i,j\}} \in \mathrm{A}$ and at least one between γ_i and γ_j belongs to A, for all $i, j \in [0, k]$. Case *(i)* arises when $\mathrm{B} = \{\alpha_{k-2|w|}\}$ (*resp.*, $\mathrm{B} = \{\alpha_{k-2|w|+1}\}$), while Case *(ii)* when $\mathrm{B} = \mathrm{Wn}_{\mathrm{o}}(\widehat{\partial}^k_{w\mathsf{L}})$. Consequently, the required property on wx immediately follows from the inductive hypothesis on w, since, if a position $\delta^{\wp}_{\{i,j\}}$ is removed from the game, also one between γ_i and γ_j is removed as well and *vice versa*. Now, let ∂ be an arbitrary game in G^k. Thanks to the topology of the games in the SCC family and to the property proved above, it easy to see that all positions in $\mathrm{X} = \{\beta_i, \gamma_i, \delta^{\wp}_{\mathrm{I}} \in \partial\}$ form a strongly connected subgame. Indeed, two positions β_i and γ_i are mutually reachable due to the two moves $(\beta_i, \gamma_i), (\gamma_i, \beta_i) \in Mv$. Moreover, two arbitrary positions γ_i and γ_j, with $i \neq j$, are mutually reachable via the positions $\delta^{\wp}_{\{i,j\}}$. Also, there are no isolated positions $\delta^{\wp}_{\{i,j\}}$. Finally, to prove that ∂ is indeed a single SCC, it remains just to show that the positions in $\mathrm{Y} = \{\alpha_i \in \partial\}$ can reach and can be reached by those in X. The first part is implied by Item 1 of Lemma 5.3.1, since every α_i has only a move to β_i, which needs to belong to ∂ in order for this to be a game. Now, due to the same observation, all positions α_i, but possibly the last one α_m with maximal index m in ∂, are reachable by β_{i+1}. Finally, α_m can be reached by γ_{m-1}, which necessarily belongs to ∂ due to Item 3 of the same lemma. □

Putting everything together, we obtain the following structural, although non asymptotic, strengthening of Theorem 5.3.1.

Theorem 5.5.1 (SCC-Decomposition Worst Case)**.** *The number of distinct recursive calls executed by the Recursive algorithm with SCC decomposition on a game with n positions is* $\Omega\left(2^{\sqrt{n/3}}\right)$ *in the worst case.*

Proof. Due to Lemma 5.5.1, the *SCC family* is an exponential worst-case family. As shown in the proof of Theorem 5.3.1, the Recursive algorithm performs at least $3(2^{\lfloor k/2 \rfloor + 1} - 1)$ calls to solve a game of $\{\partial^k\}^{\omega}_{k=1}$, and therefore, of $\{\partial^k_{\mathsf{S}}\}^{\omega}_{k=1}$. As consequence of Lemma 5.5.2, the number of calls cannot be affected by an SCC-decomposition technique, since there is an exponential number of subgames, the ones in the induced subgame tree of ∂^k_{S}, each of which forms a single SCC. Moreover, a core game ∂^k_{C} has $3(k+1)$ positions, while the number of additional positions P in the extension ∂^k_{C} is given by $\frac{3k^2+\wp}{4} + k$, which follows from Definition 5.5.1. Indeed, for every pair of positions γ_i, γ_j, with $i, j \in [0, k]$ and $i \neq j$, there is a single position $\delta^{\wp}_{\{i,j\}}$, if $i \equiv_2 j$, and two such positions, otherwise. Hence, ∂^k_{S} has $n \triangleq \frac{3k^2+\wp}{4} + 4k + 3$ positions, from which we obtain that $k = \frac{\sqrt{4(3n+7)-3\wp}-8}{3} = \frac{\lceil\sqrt{4(3n+7)-3}\rceil-8}{3} = \frac{\lceil\sqrt{12n+25}\rceil-8}{3}$. As a consequence,

the number of recursive calls is bounded from below by $3\left(2^{\left\lfloor \frac{\lceil\sqrt{12n+25}\rceil-2}{6} \right\rfloor}\right) = \Omega\left(2^{\sqrt{n/3}}\right)$. □

5.6 Dominion-Decomposition Resilient Games

A deeper analysis of the SCC family reveals that the size of the smallest dominion for player 0 in Game $\mathfrak{I}^k_{\mathsf{S}}$ (there are no dominions for player 1, being the game completely won by its opponent) is of size $k+1$. This observation, together with the fact that the game has $n = \frac{3k^2+\wp}{4} + 4k + 3 < (k+1)^2$ positions, immediately implies that the proposal of [JPZ06, JPZ08] of a brute force search for dominions of size at most $\lceil\sqrt{n}\,\rceil < k+1$ cannot help improving the solution process on these games. We can prove an even stronger result, since this kind of search cannot reduce the running time in any of the subgames of the induced tree. The reason is that the smallest dominion in each such subgame that contains at least a position in the core has size linear *w.r.t.* k, as reported by the following lemma.

Lemma 5.6.1. *For all $k \in \mathbb{N}_+$, let $\mathfrak{I}$ be a subgame in the induced subgame tree G^k of $\mathfrak{I}^k_{\mathsf{S}}$, and $\mathrm{D} \subseteq \mathfrak{I}$ the smallest dominion such that $\mathrm{D} \cap \mathfrak{I}^k_{\mathsf{C}} \neq \emptyset$. Then, $|\mathrm{D}| \geq \lfloor k/2 \rfloor + 1$.*

Proof. We start by proving that any such dominion D must necessarily contain at least a position γ_i, from some $i \in \mathbb{N}_+$. Assume, by contradiction, that it does not. Then, $\mathrm{D} \subseteq \{\alpha_i, \beta_i \in \mathfrak{I} : i \in [0,k]\} \cup \mathrm{P}$.

We can prove that D is not a game. By assumption, D must contain at least one position α_i or β_i, otherwise $\mathrm{D} \cap \mathfrak{I}^k_{\mathsf{C}} = \emptyset$. Let j be the smallest index such that $\alpha_j \in \mathrm{D}$ or $\beta_j \in \mathrm{D}$. Then, at least one of those two positions has only moves leading outside D. Hence D is not a game and, *a fortiori*, cannot be a dominion.

Therefore, D must contain at least a position γ_i. By Definition 5.5.1, γ_i is connected in $\mathfrak{I}^k_{\mathsf{S}}$ to precisely k positions $\delta^{\wp}_{\{i,j\}}$, where $j \in [0,k]$, $j \neq i$, and whose owner $\wp \equiv_2 i$ is the adversary of the owner $\overline{\wp}$ of γ_i. Let us consider the $\lfloor k/2 \rfloor$ indexes $j \in [0,k]$ such that $i \not\equiv_2 j$. For each such j, we have two cases, depending on whether γ_j belongs to $\mathfrak{I}$ or not. If it does, then so does $\delta^{\wp}_{\{i,j\}}$, since both are owned by the same player $\wp$ and are mutually connected. If it does not, then it must have been removed by some application of Item 2 of Definition 5.3.1. If the involved attractor *w.r.t.* some player $\wp'$ does not attract $\delta^{\overline{\wp}}_{\{i,j\}}$, then it cannot attract $\delta^{\wp}_{\{i,j\}}$ either, as it has no moves to γ_j. If, on the other hand, $\delta^{\overline{\wp}}_{\{i,j\}}$ gets attracted, it must belong to player $\wp'$. As a consequence, $\delta^{\wp}_{\{i,j\}}$ belongs to the opponent player $\overline{\wp'}$, in other words $\wp = \overline{\wp'}$. In either case, we conclude hat $\delta^{\wp}_{\{i,j\}}$ cannot be removed and, therefore, is still contained in $\mathfrak{I}$. In addition, all the $\lfloor k/2 \rfloor$ positions $\delta^{\wp}_{\{i,j\}}$, with $j \in [0,2k]$ and $j \not\equiv_2 i$, are mutually connected to γ_i and their owner is the opponent of the one of γ_i. It is immediate to see that if i is the

greatest index such that $\gamma_i \in \mathfrak{D}$, then it is contained in the smallest dominion D in $\mathfrak{D}$, toghether with the $\lfloor k/2 \rfloor$ positions $\delta^{\wp}_{\{i,j\}}$ connected to it. Hence, D contains at least $\lfloor k/2 \rfloor + 1$ positions.

□

The above observation allows us to obtain an exponential lower bound for the Recursive algorithm combined with memoization, SCC decomposition, and dominion decomposition techniques. Indeed, the brute-force procedure employed by the Dominion Decomposition algorithm of [JPZ06] needs at least time $\Omega\big(2^{\lfloor k/2 \rfloor + 1}\big)$ to find a dominion of size $\lfloor k/2 \rfloor + 1$.

Inspired by the Dominion Decomposition approach, in a game with n positions and p priorities, the BigStep algorithm [Sch17] tries to find a dominion of size bounded by $\pi = \sqrt[3]{pn^2}$ before each call to the internal recursive algorithm. Unlike Dominion Decomposition, however, the search is not performed via a brute-force procedure. Big Step employs, instead, a modified version of the SPM algorithm in which the search space of the possible measures evaluated during the lift computation is suitably limited by the parameter π. Moreover, this procedure, called *approximate*, does not look for an arbitrary dominion, but for one of the opposite player $\wp$ *w.r.t.* the parity of the maximal priority in the current subgame. Since player $\wp$ is also the winning player of a game $\mathfrak{D}^k_{\mathsf{S}}$ in the SCC family, the approximate SMP procedure is called *w.r.t.* player $\overline{\wp}$ so that, if a fixpoint different from $\top$ is reached, a $\wp$-dominion of the desired size exists. For $\mathfrak{D}^k_{\mathsf{S}}$, such a $\wp$-dominion actually exists, therefore a fixpoint different from $\top$ would be reached, thus making the argument in the proof of Lemma 5.4.2 not applicable. The following lemma, however, shows that even if a fixpoint is reached, the number of lifts performed is still exponential in the index of the game.

Lemma 5.6.2. *The solution time of the Big Step approximate subroutine on an element $\mathfrak{D}^k_{\mathsf{S}}$ of a worst-case family $\{\mathfrak{D}^k_{\mathsf{S}}\}^{\omega}_{k \geq 1}$ is $\Omega\Big(6^{\frac{k}{2}}\Big)$.*

Proof. The exploited SPM variant on the game $\mathfrak{D}^k_{\mathsf{S}}$ for player $\overline{\wp} = \mathsf{pr}(\alpha_k) \bmod 2 = 1 - k \bmod 2$ uses measures $m \in \mathrm{M}^{\overline{\wp}} \setminus \{\top\}$ such that $\sum_{d \in \mathsf{dom}(m)} m(d) < \pi$, where $\pi = \sqrt[3]{pn^2}$ with $p = 2(k+1)$ and $n = \frac{3k^2+\wp}{4} + 4k + 3$. Similarly to the argument in the proof of Lemma 5.4.2, every such measure is isomorphic to a $(k+1)$-tuple $\langle b_{\frac{k-\wp}{2}}, \ldots, b_0, t_{\frac{k+\wp}{2}-1}, \ldots, t_0 \rangle$ of numbers, with $b_i \in \{0,1\}$, for all $i \in [0, \frac{k-\wp}{2}]$, $t_i \in \{0,1,2\}$, for all $i \in [1, \frac{k+\wp}{2} - 1]$, and $t_0 \in [0, \frac{3k^2+\wp}{4} + k + 2]$, if $\wp = 1$, and $t_0 \in \{0,1,2\}$, otherwise. Note, indeed, that a game of the SCC family of index k contains precisely $\frac{3k^2+\wp}{4} + k + 2$ nodes with priority 0. Therefore, it is easy to observe that all measures m with $t_0 \in \{0,1,2\}$ satisfy the above restriction, since $\sum_{d \in \mathsf{dom}(m)} m(d) \leq \sum_{i=0}^{\frac{k-\wp}{2}} 1 + \sum_{i=0}^{\frac{k+\wp}{2}-1} 2 = \frac{k-\wp}{2} + 1 + 2\frac{k+\wp}{2} = \frac{3k+\wp}{2} + 1 < \pi$. Now, as noted above, the smallest $\wp$-dominion contained into $\mathfrak{D}^k_{\mathsf{S}}$ has size $k + 1 < \pi$, hence, the approximate

subroutine will find it once a fixed point $\iota_\omega = \mu\iota.\mathsf{lift}^\wp(\iota)$ of the $\mathsf{lift}^\wp$ function is reached. Due to the structure of the core family, such a fixed point maps all positions $\alpha_i, \beta_i, \gamma_i$ to the following measures:

- for all $i \in [0, \frac{k-\wp}{2}]$ and $d \in [0, 2k+1+\wp]$ with $d \equiv_2 \overline{\wp}$, it holds that $\iota_\omega(\alpha_{k-2i})(d) = \iota_\omega(\alpha_{k-2i-1})(d) = 1$, if $d \geq 2(k-i)+1+\wp$, and $\iota_\omega(\alpha_{k-2i})(d) = \iota_\omega(\alpha_{k-2i-1})(d) = 0$, otherwise; intuitively, the measures associated with the positions α_{k-2i} and α_{k-2i-1} are isomorphic to the $(k+1)$-tuple $\langle 1, \ldots, 1, 0, \ldots, 0\rangle$, where only the first $i+1$ components are set to 1, while the remaining ones to 0;

- for all $d \in [0, 2k+1+\wp]$ with $d \equiv_2 \overline{\wp}$, it holds that $\iota_\omega(\beta_k)(d) = \iota_\omega(\gamma_k)(d) = 0$; intuitively, β_k and γ_k belong to the $\wp$-dominion contained into $\mathfrak{D}^k_{\mathsf{S}}$ and no lift is ever performed on their measures;

- for all $i \in [0, \frac{k-\wp}{2}[$, it holds that $\iota_\omega(\beta_{2i+\wp}) = \iota_\omega(\gamma_{2i+\wp}) = \iota_\omega(\alpha_{2i+\wp+1})$; intuitively, the positions $\beta_{2i+\wp}$ and $\gamma_{2i+\wp}$ just propagate the measure associated with the $\alpha_{2i+\wp+1}$ position to which they are connected, since the parity of their priority is not congruent to the player $\overline{\wp}$ *w.r.t.* which the execution of the algorithm is performed;

- for all $i \in [0, \frac{k-\wp}{2}]$ and $d \in [0, 2k+1+\wp]$ with $d \equiv_2 \overline{\wp}$, it holds that $\iota_\omega(\gamma_{2i+\overline{\wp}})(d) = 1$, if $d \geq 2(k-i)+1+\wp$ or $d = 2i+\overline{\wp}$, and $\iota_\omega(\gamma_{2i+\overline{\wp}})(d) = 0$, otherwise; intuitively, the measure associated with $\gamma_{2i+\overline{\wp}}$ needs to be the smallest one greater than the one associated with $\alpha_{2i+\overline{\wp}+1}$ *w.r.t.* the truncation priority $2i+\overline{\wp}$, since the parity of the latter is congruent to the player $\overline{\wp}$;

- finally, for all $i \in [0, \frac{k-\wp}{2}]$ and $d \in [0, 2k+1+\wp]$ with $d \equiv_2 \overline{\wp}$, it holds that $\iota_\omega(\beta_{2i+\overline{\wp}})(d) = 1$, if $d \geq 2(k-i)+1+\wp$, $\iota_\omega(\beta_{2i+\overline{\wp}})(d) = 2$, if $d = 2i+\overline{\wp}$, and $\iota_\omega(\beta_{2i+\overline{\wp}})(d) = 0$, otherwise; intuitively, the measure associated with $\beta_{2i+\overline{\wp}}$ needs to be the smallest one greater than the one associated with $\gamma_{2i+\overline{\wp}}$ *w.r.t.* the truncation priority $2i+\overline{\wp}$, since the parity of the latter is congruent to the player $\overline{\wp}$.

By applying Lemma 5.4.1, we can observe that, during the computation of the lifts and before reaching the final measure $\iota_\omega(\gamma_{\overline{\wp}})$, position $\gamma_{\overline{\wp}}$ is assigned all measures in $\mathrm{M}^\wp \setminus \{\top\}$ smaller than or equal to $\iota_\omega(\gamma_{\overline{\wp}})$ that satisfy the previously described restriction. Now, it is easy to see that there are at least $2^\wp \cdot 6^{\frac{k+\wp}{2}} - 3^{\frac{k+\wp}{2}} + 1$ measures $m \in \mathrm{M}^\wp \setminus \{\top\}$ with $m(\overline{\wp}) \in \{0,1,2\}$ smaller than $\iota_\omega(\gamma_{\overline{\wp}})$. Indeed, *(i)* every such measure can be interpreted as the following natural number $3^{\frac{k+\wp}{2}} \left(\sum_{i=0}^{\frac{k-\wp}{2}} 2^i \cdot m(k+1+\wp+2i)\right) + \left(\sum_{i=0}^{\frac{k+\wp}{2}-1} 3^i \cdot m(2i+\overline{\wp})\right)$, *(ii)* every lift of $\gamma_{\overline{\wp}}$ increments the value associated with its measure just

by one, and *(iii)* the value of $\iota_\omega(\gamma_{\overline{\wp}})$ is precisely $3^{\frac{k+\wp}{2}}\left(\sum_{i=0}^{\frac{k-\wp}{2}} 2^i\right) + 1 = 3^{\frac{k+\wp}{2}}\left(2^{\frac{k-\wp}{2}+1} - 1\right) + 1 = 2^{\overline{\wp}} \cdot 6^{\frac{k+\wp}{2}} - 3^{\frac{k+\wp}{2}} + 1 = \Omega\left(6^{\frac{k}{2}}\right)$. □

As a consequence, none of the dominion decomposition approaches, combined with memoization and SCC decomposition, can efficiently solve the SCC family.

Corollary 5.6.1 (Exponential Dominion-Decomposition Worst Case)**.** *The solution time of the Recursive algorithm with memoization, SCC decomposition, and dominion decomposition on a game with n positions is* $\Omega\left(2^{\Theta(\sqrt{n})}\right)$ *in the worst case.*

Chapter 6

Priority Promotion

In the current chapter we describe a new efficient solution approach for parity games, obtained analyzing the structure of the worst-case family presented in the previous chapter. The proposed technique, together with all its refinements described in the next chapters, can easily solve, indeed, every element of this family in polynomial time.

This work has been presented at the

- 28th International Conference on Computer Aided Verification (CAV'16), held in July 17-23, 2016, in Toronto, Ontario, Canada, with the title of "Solving Parity Games via Priority Promotion" and published on Volume 9780 (Part II) of LNCS, pages 270-290. Springer, 2016,
- 18th Italian Conference on Theoretical Computer Science (ICTCS'17), held in September 26-29, 2017, in Naples, Italy, with the title of "The Priority Promotion Approach to Parity Games", and
- 2nd Winter School in Engineering and Computer Science on Formal Verification (CSE 2), held in Dicembre 17-21, 2017, in Jerusalem, Israel, with the title of "Solving parity games via priority promotion".

The content of the chapter is based on the journal version of [BDM16c] published in Formal Methods in System Design [BDM18b].

6.1 Abstract

We consider *parity games*, a special form of two-player infinite-duration games on numerically labeled graphs, whose winning condition requires that the maximal value of a label occurring infinitely often during a play be of some specific parity. The problem of identifying the corresponding winning regions has a rather intriguing status from a complexity theoretic viewpoint, since it belongs to the class UPTime ∩ CoUPTime, and still open is the question whether it can be solved in polynomial time. Parity games also have great practical interest, as they arise in many fields of theoretical computer science, most notably logic, automata theory, and formal verification. Our main contribution is a new algorithm for solving parity games, based on the notions of *quasi dominion* and *priority promotion*. A quasi dominion Q for player $\wp \in \{0,1\}$, called a *quasi $\wp$-dominion*, is a set of positions from each of which player $\wp$ can enforce a winning play that never leaves the region, unless one of the following two conditions holds: *(i)* the opponent $\overline{\wp}$ can escape from Q or *(ii)* the only choice for player $\wp$ itself is to exit from Q (*i.e.*, no move from a position of $\wp$ remains in Q). Quasi dominions can be ordered by assigning to each of them a priority corresponding to an under-approximation of the best value the opponent can be forced to visit along any play exiting from it. A crucial property is that, under suitable and easy to check assumptions, a higher priority quasi $\wp$-dominion Q_1 and a lower priority one Q_2, can be merged into a single quasi $\wp$-dominion of the higher priority, thus improving the approximation for Q_2. For this reason we call this merging operation a priority promotion of Q_2 to Q_1. The underlying idea of our approach is to iteratively enlarge quasi $\wp$-dominions, by performing sequences of promotions, until an $\wp$-dominion is obtained.

We prove soundness and completeness of the algorithm. Moreover, we provide an upper bound $\mathrm{O}\left(kn\left(\frac{\mathsf{e}n}{k-1}\right)^{k-1}\right)$ on the time complexity, where e is Euler's number, and a bound $\mathrm{O}(n \cdot \log k)$ on the memory requirements. Experimental results, comparing our algorithm with the state of the art solvers, also show that the proposed approach performs very well in practice, most often significantly better than existing ones, on both random games and benchmark families proposed in the literature.

6.2 A New Idea

A solution for a parity game $\mathcal{D} = \langle \mathcal{A}, \mathrm{Pr}, \mathsf{pr} \rangle \in \mathcal{P}$ over an arena $\mathcal{A} = \langle \mathrm{Ps}^0, \mathrm{Ps}^1, Mv \rangle$ can trivially be obtained by iteratively computing dominions of some player, namely sets of positions from which that player has a strategy to win the game. Once an α-dominion D for player $\wp \in \{0,1\}$ is found, its $\wp$-attractor $\mathsf{atr}^{\wp}_{\mathcal{D}}(\mathrm{D})$ gives an $\wp$-maximal dominion containing D. In other words, $\wp$ cannot force any

position outside D to enter this set. The subgame $\partial \setminus \mathsf{atr}^{\wp}_{\partial}(\mathrm{D})$ can then be solved by iterating the process. This procedure is reported in Algorithm 5. The crucial problem to address, therefore, consists in computing a dominion for some player in the game. The difficulty here is that, in general, no unique priority exists that satisfies the winning condition for a player along all the plays inside the dominion we are looking for. In fact, that value depends on the strategy chosen by the opponent. Our solution to this problem is to proceed in a bottom-up fashion, starting from a weaker notion of $\wp$-dominion, called *quasi $\wp$-dominion*. Then, we compose quasi $\wp$-dominions until we obtain an $\wp$-dominion. Intuitively, a quasi $\wp$-dominion is a set of positions on which player $\wp$ has a strategy whose induced plays either remain in the set forever and are winning for $\wp$, or can exit from it passing through a specific set of positions, *i.e.*, the escapes of the set itself. This notion is formalized by the following definition.

Algorithm 5: Parity Game Solver.

```
signature sol_Γ : P →_∂ (2^{Ps_∂} × 2^{Ps_∂})
function sol_Γ[∂]
    if Ps_∂ = ∅ then
        return (∅, ∅)
    else
        (R, ℘) ← src_Γ(∂)
        R* ← atr^℘_∂(R)
        (W_o', W_1') ← sol_Γ[∂ \ R*]
        (W_℘, W_¬℘) ← (W_℘' ∪ R*, W_¬℘')
        return (W_o, W_1)
```

Algorithm 6: The Searcher.

```
signature src_Γ : P →_∂ QD+_∂
function src_Γ(∂)
    return src_{Γ(∂)}(⊤_{Γ(∂)})

signature src_D : S_D → QD+_{∂_D}
function src_D(s)
    (Q, ℘) ← ℜ_D(s)
    if (Q, ℘) ∈ QD+_{∂_D} then
        return (Q, ℘)
    else
        return src_D(s↓_D(Q, ℘))
```

Definition 6.2.1 (Quasi Dominion)**.** *Let $\partial \in \mathcal{P}$ be a game and $\wp \in \{0,1\}$ a player. A non-empty set of positions* $\mathrm{Q} \subseteq \mathrm{Ps}_{\partial}$ *is a* quasi $\wp$-dominion *in ∂ if there exists an $\wp$-strategy $\sigma_{\wp} \in \mathrm{Str}^{\wp}_{\partial}(\mathrm{Q})$ such that, for all $\overline{\wp}$-strategies $\sigma_{\overline{\wp}} \in \mathrm{Str}^{\overline{\wp}}_{\partial}(\mathrm{Q})$, with $\mathsf{int}^{\overline{\wp}}_{\partial}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\wp}})$, and positions $v \in \mathrm{Q}$, the induced play $\pi = \mathsf{play}_{\partial}((\sigma_{\mathrm{o}}, \sigma_{\mathrm{1}}), v)$ satisfies $\mathsf{pr}_{\partial}(\pi) \equiv_2 \wp$, if π is infinite, and $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{Q})$, otherwise.*

It is important to observe that the additional requirement that the opponent strategies be defined on all interior positions, formally $\mathsf{int}^{\overline{\wp}}_{\partial}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\wp}})$, discards those strategies in which the opponent deliberately chooses to forfeit the play, by declining to take any move at some of its positions.

We say that a quasi $\wp$-dominion Q is *$\wp$-open* (*resp.*, *$\wp$-closed*) if $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{Q}) \neq \emptyset$ (*resp.*, $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{Q}) = \emptyset$).

In other words, in a closed quasi $\wp$-dominion, player $\wp$ has a strategy whose induced plays are all infinite and winning. Hence, when closed, a quasi $\wp$-dominion is a dominion for $\wp$ in $\mathfrak{D}$. The set of pairs $(\mathrm{Q}, \wp) \in 2^{\mathrm{Ps}_{\mathfrak{D}}} \times \{0, 1\}$, where Q is a quasi $\wp$-dominion, is denoted by $\mathrm{QD}_{\mathfrak{D}}$, and is partitioned into the sets $\mathrm{QD}^{-}_{\mathfrak{D}}$ and $\mathrm{QD}^{+}_{\mathfrak{D}}$ of open and closed quasi $\wp$-dominion pairs, respectively. As an example, consider Figure 6.1. It is easy to see that the sets of positions $\{a\}$ and $\{b, c, d\}$ form two open quasi 0-dominion in the game $\mathfrak{D}$. Similarly, the sets of positions $\{c, d\}$ and $\{e\}$ are open quasi 1-dominion. Finally, the entire game $\mathfrak{D}$ is a closed quasi 0-dominion, or simply a 0-dominion, where the 0-strategy correspond to the bold arrows.

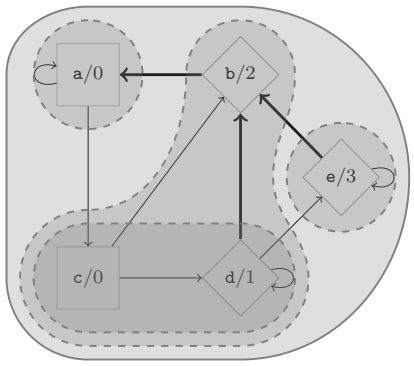


Figure 6.1: A simple game $\mathfrak{D}$.

An expert reader might note that quasi $\wp$-dominions are loosely related with the concept of *snares*, introduced in [Fea10] and used there for completely different purposes, namely to speed up the convergence of standard strategy improvement algorithms.

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

Definition 6.2.2 (Dominion Space)**.** *A* dominion space *for a game* $\mathfrak{D} \in \mathcal{P}$ *is a tuple* $\mathcal{D} \triangleq \langle \mathfrak{D}, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where* (1) $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$ *is a well-founded partial order* w.r.t. $\prec \subset \mathrm{S} \times \mathrm{S}$ *with distinguished element* $\top \in \mathrm{S}$*,* (2) $\succ \subseteq \mathcal{S} \times \mathrm{QD}^{-}_{\mathfrak{D}}$ *is the* compatibility relation*,* (3) $\Re : \mathrm{S} \to \mathrm{QD}_{\mathfrak{D}}$ *is the* query function *mapping each element* $s \in \mathrm{S}$ *to a quasi dominion pair* $(\mathrm{Q}, \wp) \triangleq \Re(s) \in \mathrm{QD}_{\mathfrak{D}}$ *such that, if* $(\mathrm{Q}, \wp) \in \mathrm{QD}^{-}_{\mathfrak{D}}$ *then* $s \succ (\mathrm{Q}, \wp)$*, and* (4) $\downarrow : \succ \to \mathrm{S}$ *is the* successor function *mapping each pair* $(s, (\mathrm{Q}, \wp)) \in \succ$ *to the element* $s^{\star} \triangleq s{\downarrow}(\mathrm{Q}, \wp) \in \mathrm{S}$ *with* $s^{\star} \prec s$*.*

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

Different dominion spaces can be associated to the same game. Therefore, in the rest of this section, we shall simply assume a function Γ mapping every game ∂ to a dominion space $\Gamma(\partial)$. Given the top element of $\mathcal{D} = \Gamma(\partial)$, Algorithm 6 searches for a dominion of either one of the two players by querying the current state s for a region pair $(\mathrm{Q}, \wp)$. If this is closed in ∂, it is returned as an $\wp$-dominion. Otherwise, a successor state $s\downarrow_{\mathcal{D}}(\mathrm{Q}, \wp)$ is computed and the search proceeds recursively from it. Clearly, since the partial order is well-founded, termination of the $\mathsf{src}_{\mathcal{D}}$ procedure is guaranteed. The total number of recursive calls is, therefore, the execution depth $\mathsf{d}_{\mathcal{D}}(n, m, k)$ of the dominion space $\mathcal{D}$, where n, m, and k are the number of positions, moves, and priorities, respectively. Hence, $\mathsf{src}_{\mathcal{D}}$ runs in time $\mathrm{O}(\mathsf{d}_{\mathcal{D}}(n, m, k) \cdot (\mathsf{T}_{\Re}(n, m) + \mathsf{T}_{\downarrow}(n, m))$, where $\mathsf{T}_{\Re}(n, m)$ and $\mathsf{T}_{\downarrow}(n, m)$ denote the time needed by the query and successor functions, respectively. Thus, the total time to solve a game is $\mathrm{O}(m + n \cdot \mathsf{d}_{\mathcal{D}}(n, m, k) \cdot (\mathsf{T}_{\Re}(n, m) + \mathsf{T}_{\downarrow}(n, m)))$. Since the query and successor functions of the dominion space considered in the rest of the paper can be computed in linear time *w.r.t.* both n and m, the whole procedure terminates in time $\mathrm{O}(n \cdot (n + m) \cdot \mathsf{d}_{\mathcal{D}}(n, m, k))$. As to the space requirements, observe that $\mathsf{src}_{\mathcal{D}}$ is a tail recursive algorithm. Hence, the upper bound on memory only depends on the space needed to encode the states of a dominion space, namely $\mathrm{O}(\log \|\mathcal{D}\|)$, where $\|\mathcal{D}\|$ is the size of the partial order $\mathcal{S}$ associated with $\mathcal{D}$.

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

6.3 Priority Promotion

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

Definition 6.3.1 (Region)**.** *A quasi $\wp$-dominion* R *is an $\wp$-region if* $\mathsf{pr}(\partial) \equiv_2 \wp$ *and all the positions in* $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{R})$ *have priority* $\mathsf{pr}(\partial)$*,* i.e. $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\partial}(\mathsf{pr}(\partial))$.

As a consequence of the above definition, if the opponent $\overline{\wp}$ can escape from an $\wp$-region, it must visit a position with the highest priority in the region, which is of parity $\wp$. Similarly to the case of quasi dominions, we shall denote with Rg_{∂} the set of region pairs in ∂ and with $\mathrm{Rg}_{\partial}^{-}$ and $\mathrm{Rg}_{\partial}^{+}$ the sets of open and closed region pairs, respectively. A closed $\wp$-region is clearly an $\wp$-dominion. As an example, consider again Figure 6.1. The singleton $\{e\}$ is the unique 1-region in the entire game ∂, which is also open, while $\{b\}$ is a non-maximal 0-region in the subgame $\partial \setminus \{e\}$ where e is removed. Finally, the set $\{b, c, d\}$ is a maximal open 0-region in the same subgame.

At this point, we have all the tools to explain the crucial steps underlying the search procedure. Open regions are not winning, as the opponent can force plays exiting from them. Therefore, in order to build a dominion starting from open regions, we look for a suitable sequence of regions that can be merged together until a closed one is found. Obviously, the merging operation needs to be applied only to regions belonging to the same player, in such a way that the resulting set of position is still a region of that player. To this end, a mechanism is proposed, where an $\wp$-region R in some game ∂ and an $\wp$-dominion D in a subgame of ∂ not containing R itself are merged together, if the only moves exiting from $\overline{\wp}$-positions of D in the entire game lead to higher priority $\wp$-regions and R has the lowest priority among them. As we shall see, this ensures that the new region $\mathrm{R}^{\star} \triangleq \mathrm{R} \cup \mathrm{D}$ has the same associated priority as R. This merging operation, based on the following proposition, is called *promotion* of the lower region to the higher one.

Proposition 6.3.1 (Region Merging)**.** *Let $\partial \in \mathcal{P}$ be a game, $\mathrm{R} \subseteq \mathrm{Ps}_{\partial}$ an $\wp$-region, and $\mathrm{D} \subseteq \mathrm{Ps}_{\partial \setminus \mathrm{R}}$ an $\wp$-dominion in the subgame $\partial \setminus \mathrm{R}$. Then, $\mathrm{R}^{\star} \triangleq \mathrm{R} \cup \mathrm{D}$ is an $\wp$-region in ∂. Moreover, if both R and D are $\wp$-maximal in ∂ and $\partial \setminus \mathrm{R}$, respectively, then $\mathrm{R}^{\star}$ is $\wp$-maximal in ∂ as well.*

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

Condition *(ii)* immediately follows from the assumption that R is an $\wp$-region in ∂. To show that also Condition *(iii)* holds, we observe that, since D is an $\wp$-dominion in $\partial \setminus \mathrm{R}$, the only possible moves exiting from $\overline{\wp}$-positions of D in game ∂ must lead to R, *i.e.*, $\mathsf{esc}_{\partial}^{\overline{\wp}}(\mathrm{D}) \subseteq \mathrm{R}$. Hence, the only escaping

positions of $\mathrm{R}^\star$, if any, must belong to R, *i.e.* $\mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}^\star) \subseteq \mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R})$. Since R is an $\wp$-region in $\mathfrak{D}$, it hods that $\mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathfrak{D}}(\mathsf{pr}(\mathfrak{D}))$. By transitivity, we conclude that $\mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}^\star) \subseteq \mathsf{pr}^{-1}_{\mathfrak{D}}(\mathsf{pr}(\mathfrak{D}))$.

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

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

Let now π be a play compatible with $\sigma_{\mathrm{R}^\star}$. If π is an infinite play, then it remains forever in $\mathrm{R}^\star$ and we have three possible cases. If π eventually remains forever in D, then it is clearly winning for $\wp$, since $\sigma_{\mathrm{R}^\star}$ coincides with σ_{D} on all the positions in D. Similarly, if π eventually remains forever in R, then it is also winning for $\wp$, as $\sigma_{\mathrm{R}^\star}$ coincides with σ_{R} on all the positions in R. If, on the other hand, π passes infinitely often through both R and D, it necessarily visits infinitely often an escaping position in $\mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\mathfrak{D}}(\mathsf{pr}(\mathfrak{D}))$, which has the maximal priority in $\mathfrak{D}$ and is of parity $\wp$. Hence, the parity of the maximal priority visited infinitely often along π is $\wp$ and π is winning for player $\wp$. Finally, if π is a finite play, then it must end at some escaping position of R from where $\wp$ cannot force to move to a position still in $\mathrm{R}^\star$, *i.e.*, it must end in a position of the set $\mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}) \setminus \mathsf{pre}^{\wp}_{\mathfrak{D}}(\mathrm{D}) = \mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}^\star)$. Therefore, $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\wp}}_{\mathfrak{D}}(\mathrm{R}^\star)$. We can then conclude that $\mathrm{R}^\star$ also satisfies Condition *(i)*.

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

During the search, we keep track of the computed regions by means of an auxiliary priority function $\mathsf{r} \in \mathbb{P}_{\mathfrak{D}} \triangleq \mathrm{Ps}_{\mathfrak{D}} \to \mathrm{Pr}_{\mathfrak{D}}$, called *region function*, which formalizes the intuitive notion of priority of a region described above. Initially, the region function coincides with the priority function $\mathsf{pr}_{\mathfrak{D}}$ of the entire game $\mathfrak{D}$. Priorities are considered starting from the highest one. A region of the same parity $\wp \in \{0, 1\}$ of the priority p under consideration is extracted from the region function, by collecting the set of positions $\mathsf{r}^{-1}(p)$. Then, its attractor $\mathrm{R} \triangleq \mathsf{atr}^{\wp}_{\mathfrak{D}^\star}(\mathsf{r}^{-1}(p))$ is computed *w.r.t.* the subgame $\mathfrak{D}^\star$, which is derived from

∂ by removing the regions with priority higher than p. The resulting set forms an $\wp$-maximal set of positions from which the corresponding player can force a visit to positions with priority p. This first phase is called *region extension*. If the $\wp$-region R is open in $\partial^\star$, we proceed and process the next priority. In this case, we set the priority of the newly computed region to p. Otherwise, one of two situations may arise. Either R is closed in the whole game ∂ or the only $\overline{\wp}$-moves exiting from R lead to higher regions of the same parity. In the former case, R is a $\wp$-dominion in the entire game and the search stops. In the latter case, R is only an $\wp$-dominion in the subgame $\partial^\star$, and a promotion of R to a higher region $\mathrm{R}^\natural$ can be performed, according to Proposition 6.3.1. The search, then, restarts from the priority of $\mathrm{R}^\natural$, after resetting to the original priorities in pr_∂ all the positions of the lower priority regions. The region $\mathrm{R}^\star$ resulting from the union of $\mathrm{R}^\natural$ and R will then be reprocessed and, possibly, extended in order to make it $\wp$-maximal. If R can be promoted to more than one region, the one with the lowest priority is chosen, so as to ensure the correctness of the merging operation. Due to the property of maximality, no $\overline{\wp}$-moves from R to higher priority $\overline{\wp}$-regions exist. Therefore, only regions of the same parity are considered in the promotion step. The correctness of region extension operation above, the remaining fundamental step in the proposed approach, is formalized by the following proposition.

Proposition 6.3.2 (Region Extension)**.** *Let $\partial \in \mathcal{P}$ be a game and $\mathrm{R}^\star \subseteq \mathrm{Ps}_\partial$ an $\wp$-region in ∂. Then, $\mathrm{R} \triangleq \mathsf{atr}^\wp_\partial(\mathrm{R}^\star)$ is an $\wp$-maximal $\wp$-region in ∂.*

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

Figure 6.2 and Table 6.1 illustrate the search procedure on an example game, where diamond shaped positions belong to player 0 and square shaped ones to the opponent 1. Player 0 wins from every position, hence the 0-region containing all the positions is a 0-dominion in this case. Each cell of the table contains a computed region. A downward arrow denotes a region that is open in the subgame where it is computed, while an upward arrow means that the region gets to be promoted to the priority in the

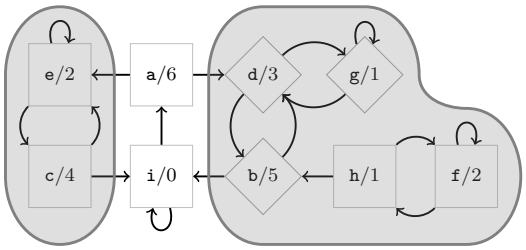


Figure 6.2: Running example.

subscript. The index of each row corresponds to the priority of the region. Following the idea sketched above, the first region obtained is the single-position 0-region $\{\mathtt{a}\}$, which is open because of the two moves leading to `d` and `e`. At priority 5, the open 1-region $\{\mathtt{b},\mathtt{f},\mathtt{h}\}$ is formed by attracting both `f` and `h` to `b`, which is open in the subgame where $\{\mathtt{a}\}$ is removed. Similarly, the 0-region $\{\mathtt{c}\}$ at priority 4 and the 1-region $\{\mathtt{d}\}$ at priority 3 are open, once removed $\{\mathtt{a},\mathtt{b},\mathtt{f},\mathtt{h}\}$ and $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{f},\mathtt{h}\}$, respectively, from the game. At priority 2, the 0-region $\{\mathtt{e}\}$ is closed in the corresponding subgame. However, it is not closed in the whole game, since it has a move leading to `c`, *i.e.*, to region 4. A promotion of $\{\mathtt{e}\}$ to 4 is then performed, resulting in the new 0-region $\{\mathtt{c},\mathtt{e}\}$. The search resumes at the corresponding priority and, after computing the extension of such a region via the attractor, we obtain that it is still open in the corresponding subgame. Consequently, the 1-region of priority 3 is recomputed and, then, priority 1 is processed to build the 1-region $\{\mathtt{g}\}$. The latter is closed in the associated subgame, but not in the original game, because of a move leading to position `d`. Hence, another promotion is performed, leading to closed region in Row 3 and Column 3, which in turn triggers a promotion to 5. Observe that every

	1	2	3	4	5	6	7
6	a↓	···	···	···	···	a,b,d,g,i↓	···
5	b,f,h↓	···	···	b,d,f,g,h↓	···		
4	c↓	c,e↓	···	c↓	c,e↓	c↓	c,e,f,h$\uparrow_6$
3	d↓	d↓	d,g$\uparrow_5$				
2	e$\uparrow_4$			e$\uparrow_4$		e, f, h$\uparrow_4$	
1		g$\uparrow_3$					
0					i$\uparrow_6$		

Table 6.1: PP simulation.

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

Note that, the positions in 0-region $\{\mathtt{c}, \mathtt{e}\}$ are reset to their initial priorities, when 1-region $\{\mathtt{d}, \mathtt{g}\}$ in Column 3 is promoted to 5. Similarly, when 0-region $\{\mathtt{i}\}$ in Column 5 is promoted to 6, the priorities of the positions in both regions $\{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and $\{\mathtt{c}, \mathtt{e}\}$, highlighted by the gray areas, are reset. This is actually necessary for correctness, at least in general. In fact, if region $\{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$ were not reset, the promotion of $\{\mathtt{i}\}$ to 6, which also attracts $\mathtt{b}$, $\mathtt{d}$, and $\mathtt{g}$, would leave $\{\mathtt{f}, \mathtt{h}\}$ as a 1-region of priority 5. However, according to Definition 6.3.1, this is not a 1-region. Even worse, it would also be considered a closed 1-region in the entire game, without being a 1-dominion, since it is actually an open 0-region. This shows that, in principle, promotions to an higher priority require the reset of previously built regions of lower priorities.

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

6.3.1 PP Dominion Space

In order to define the dominion space induced by the *priority-promotion mechanism* (PP, for short), we need to introduce some additional notation. Given a priority function $\mathsf{r} \in \mathbb{P}_{\mathcal{D}}$ and a priority $p \in \mathrm{Pr}$, we denote by $\mathsf{r}^{(\geq p)}$ (*resp.*, $\mathsf{r}^{(>p)}$ and $\mathsf{r}^{(<p)}$) the function obtained by restricting the domain of r to the positions with priority greater than or equal to p (*resp.*, greater than and lower than p). Formally, $\mathsf{r}^{(\geq p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) \geq p\}$ (*resp.*, $\mathsf{r}^{(>p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) > p\}$ and $\mathsf{r}^{(<p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathsf{dom}(\mathsf{r}) : \mathsf{r}(v) < p\}$). By $\mathcal{D}_{\mathsf{r}}^{\leq p}$ we denote the largest subgame contained in the structure $\mathcal{D} \setminus \mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$, which is obtained by removing from $\mathcal{D}$ all the positions in the domain of $\mathsf{r}^{(>p)}$. A priority function $\mathsf{r} \in \mathbb{R}_{\mathcal{D}} \subseteq \mathbb{P}_{\mathcal{D}}$ in $\mathcal{D}$ is a *region function* iff, for all priorities $q \in \mathsf{rng}(\mathsf{r})$ with $\wp \triangleq q \bmod 2$, it holds that $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}_{\mathsf{r}}^{\leq q}}$ is an $\wp$-region in the subgame $\mathcal{D}_{\mathsf{r}}^{\leq q}$, if non-empty. In addition, we say that r is *maximal above* $p \in \mathrm{Pr}$ iff, for all $q \in \mathsf{rng}(\mathsf{r})$ with $q > p$, we have that $\mathsf{r}^{-1}(q)$ is $\wp$-maximal in $\mathcal{D}_{\mathsf{r}}^{\leq q}$ with $\wp \triangleq q \bmod 2$.

To account for the current status of the search of a dominion, the states s of the corresponding

dominion space need to contain the current region function r and the current priority p reached by the search in $\mathfrak{D}$. To each of such states $s \triangleq (\mathsf{r}, p)$, we then associate the *subgame at* s defined as $\mathfrak{D}_s \triangleq \mathfrak{D}_{\mathsf{r}}^{\leq p}$, representing the portion of the original game that still has to be processed. For instance, the state s corresponding to Row 4 Column 4 in Table 6.1 is $(\mathsf{r}, 4)$, where the region function r is such that $\mathsf{r}(\mathtt{a}) = 6$, $\mathsf{r}(x) = 5$, for $x \in \{\mathtt{b}, \mathtt{d}, \mathtt{f}, \mathtt{g}, \mathtt{h}\}$, and $\mathsf{r}(x) = \mathsf{pr}(x)$, for $x \in \{\mathtt{c}, \mathtt{e}, \mathtt{i}\}$. In addition, the subgame $\mathfrak{D}_s$ of s only contains the positions $\mathtt{c}$, $\mathtt{e}$, and $\mathtt{i}$.

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

Definition 6.3.2 (State Space)**.** *A* state space *is a tuple* $\mathcal{S}_{\mathfrak{D}} \triangleq \langle \mathrm{S}_{\mathfrak{D}}, \top_{\mathfrak{D}}, \prec_{\mathfrak{D}} \rangle$*, where its components are defined as prescribed in the following:*

1. $\mathrm{S}_{\mathfrak{D}} \subseteq \mathbb{R}_{\mathfrak{D}} \times \mathrm{Pr}_{\mathfrak{D}}$ *is the set of all pairs* $s \triangleq (\mathsf{r}, p)$*, called* states*, composed of a region function* $\mathsf{r} \in \mathbb{R}_{\mathfrak{D}}$ *and a priority* $p \in \mathrm{Pr}_{\mathfrak{D}}$ *such that* (a) r *is maximal above* p*,* (b) $p \in \mathsf{rng}(\mathsf{r})$*, and* (c) $\mathsf{r}^{(<p)} \subseteq \mathsf{pr}_{\mathfrak{D}}{}^{(<p)}$*;*

2. $\top_{\mathfrak{D}} \triangleq (\mathsf{pr}_{\mathfrak{D}}, \mathsf{pr}(\mathfrak{D}))$*;*

3. *for any two states* $s_1 \triangleq (\mathsf{r}_1, p_1), s_2 \triangleq (\mathsf{r}_2, p_2) \in \mathrm{S}_{\mathfrak{D}}$*, it holds that* $s_1 \prec_{\mathfrak{D}} s_2$ *iff either* (a) *there exists a priority* $q \in \mathsf{rng}(\mathsf{r}_1)$ *with* $q \geq p_1$ *such that* (a.i) $\mathsf{r}_1{}^{(>q)} = \mathsf{r}_2{}^{(>q)}$ *and* (a.ii) $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$*, or* (b) *both* (b.i) $\mathsf{r}_1 = \mathsf{r}_2$ *and* (b.ii) $p_1 < p_2$ *hold.*

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

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

r. These three accessory properties ensure that the successor function is always able to cast R inside the current region function r and obtain a new state.

Definition 6.3.3 (Compatibility Relation)**.** *An open quasi-dominion pair* $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}_{\partial}$ *is* compatible *with a state* $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_{\partial}$*, in symbols* $s \succ_{\partial} (\mathrm{R}, \wp)$*, iff* (1) $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\partial_s}$ *and* (2) *if* R *is* $\wp$*-open in* ∂_s *then* (2.a) R *is* $\wp$*-maximal in* ∂_s *and* (2.b) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$.

Algorithm 7 provides a possible implementation for the query function compatible with the priority-promotion mechanism. Let $s \triangleq (\mathsf{r}, p)$ be the current state. Line 1 simply computes the parity $\wp$ of the priority to process in that state. Line 2, instead, computes in game ∂_s the attractor *w.r.t.* player $\wp$ of the region contained in r at the current priority p. The resulting set R is, according to Proposition 6.3.2, an $\wp$-maximal $\wp$-region in ∂_s containing $\mathsf{r}^{-1}(p)$.

Algorithm 7: Query Function.

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

Before continuing with the description of the implementation of the successor function, we need to introduce the notion of *best escape priority* for player $\overline{\wp}$ *w.r.t.* an $\wp$-region R of the subgame ∂_s and a region function r in the whole game ∂. Informally, such a value represents the best priority associated with an $\wp$-region contained in r and reachable by $\overline{\wp}$ when escaping from R. To formalize this concept, let $I \triangleq Mv_{\partial} \cap ((\mathrm{R} \cap \mathrm{Ps}^{\overline{\wp}}_{\partial}) \times (\mathsf{dom}(\mathsf{r}) \setminus \mathrm{R}))$ be the *interface relation* between R and r, *i.e.*, the set of $\overline{\wp}$-moves exiting from R and reaching some position within a region recorded in r. Then, $\mathsf{bep}^{\overline{\wp}}_{\partial}(\mathrm{R}, \mathsf{r})$ is set to the minimal priority among those regions containing positions reachable by a move in I. Formally, $\mathsf{bep}^{\overline{\wp}}_{\partial}(\mathrm{R}, \mathsf{r}) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r}{\restriction}\mathsf{rng}(I)))$. Note that, if R is a closed $\wp$-region in ∂_s, then $\mathsf{bep}^{\overline{\wp}}_{\partial}(\mathrm{R}, \mathsf{r})$ is necessarily of parity $\wp$ and greater than the priority p of R. This property immediately follows from the maximality of r above p in any state of the dominion space. Indeed, no move of an $\overline{\wp}$-position can lead to a $\overline{\wp}$-maximal $\overline{\wp}$-region. For instance, in the example of Figure 6.2, for 0-region $\mathrm{R} = \{\mathsf{e}, \mathsf{f}, \mathsf{h}\}$ with priority equal to 2 in column 6, we have that $I = \{(\mathsf{e}, \mathsf{c}), (\mathsf{h}, \mathsf{b})\}$ and $\mathsf{r}{\restriction}\mathsf{rng}(I) = \{(\mathsf{c}, 4), (\mathsf{b}, 6)\}$. Hence, $\mathsf{bep}^{1}_{\partial}(\mathrm{R}, \mathsf{r}) = 4$.

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

Algorithm 8 implements the successor function informally described at the beginning of the section.

Given the current state s and a compatible region pair $(\mathrm{R}, \wp)$ open in the whole game as inputs, it produces a successor state $s^\star \triangleq (\mathsf{r}^\star, p^\star)$ in the dominion space. It first checks whether R is open also in the subgame ∂_s (Line 1). If this is the case, it assigns priority p to region R and stores it in the new region function $\mathsf{r}^\star$ (Line 2). The new current priority $p^\star$ is, then, computed as the highest priority lower than p in $\mathsf{r}^\star$ (Line 3). If, on the other hand, R is closed in ∂_s, a promotion merging R with some other $\wp$-region contained in r is required. The next priority $p^\star$ is set to the bep of R for player $\overline{\wp}$ in the entire game ∂ *w.r.t.* r (Line 4). Region R is, then, promoted to priority $p^\star$ and all the priorities below $p^\star$ in the current region function r are reset (Line 5). The correctness of this last operation follows from Proposition 6.3.1.

As already observed in Section 6.2, a dominion space, together with Algorithm 6, provides a sound and complete solution procedure. The following theorem states that the priority-promotion mechanism presented above is indeed a dominion space. For the sake of readability, the proof is provided in the appendix.

Algorithm 8: Successor Function.

```
   signature ↓_∂ : ≻_∂ → Δ_∂ × Pr_∂
   function s↓_∂ (R, ℘)
       let (r, p) = s in
1          if (R, ℘) ∈ Rg⁻_{∂_s} then
2              r* ← r[R ↦ p]
3              p* ← max(rng(r*^(<p)))
           else
4              p* ← bep^{℘̄}_∂(R, r)
5              r* ← pr_∂ ⊎ r^(≥p*)[R ↦ p*]
6      return (r*, p*)
```

Theorem 6.3.1 (Dominion Space)**.** *For a game ∂, the* structure $\mathcal{D}_\partial \triangleq \langle \partial, \mathcal{S}_\partial, \succ_\partial, \Re_\partial, \downarrow_\partial \rangle$*, where $\mathcal{S}_\partial$ is given in Definition 6.3.2, $\succ_\partial$ is the relation of Definition 6.3.3, and $\Re_\partial$ and $\downarrow_\partial$ are the functions computed by Algorithms 7 and 8 is a dominion space.*

To prove the above theorem, we have to show that the three components $\mathcal{S}_\partial$, $\Re_\partial$, and $\downarrow_\partial$ of the structure $\mathcal{D}_\partial$ satisfy the properties required by Definition 6.2.2 of dominion space. We do this through the Lemmas 6.3.1, 6.3.2, and 6.3.3.

Lemma 6.3.1 (State Space)**.** *The* PP *state space $\mathcal{S}_\partial = \langle \mathrm{S}_\partial, \top_\partial, \prec_\partial \rangle$ for a game $\partial \in \mathcal{P}$ is a well-founded partial order* w.r.t. $\prec_\partial$ *with designated element $\top_\partial \in \mathrm{S}_\partial$.*

Proof. Since S_∂ is a finite set, to show that $\prec_\partial$ is a well-founded partial order on S_∂, it is enough to prove that it is simply a strict partial order on the same set, *i.e.*, an irreflexive and transitive relation.

For the irreflexive property, by Item 3 of Definition 6.3.2, it is immediate to see that $s \not\prec_\partial s$, for all states $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_\partial$, since neither there exists a priority $q \in \mathsf{rng}(\mathsf{r})$ such that $\mathsf{r}^{-1}(q) \subset \mathsf{r}^{-1}(q)$ nor $p < p$.

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

- Item 3.a for both $s_1 \prec_\mathfrak{D} s_2$ and $s_2 \prec_\mathfrak{D} s_3$: there exist two priorities $q_1 \in \mathsf{rng}(\mathsf{r}_1)$ and $q_2 \in \mathsf{rng}(\mathsf{r}_2)$ with $q_1 \geq p_1$ such that ${\mathsf{r}_1}^{(>q_1)} = {\mathsf{r}_2}^{(>q_1)}$, ${\mathsf{r}_2}^{(>q_2)} = {\mathsf{r}_3}^{(>q_2)}$, $\mathsf{r}_2^{-1}(q_1) \subset \mathsf{r}_1^{-1}(q_1)$, and $\mathsf{r}_3^{-1}(q_2) \subset \mathsf{r}_2^{-1}(q_2)$. Let $q \triangleq \mathsf{max}\{q_1, q_2\} \geq p_1$. If $q = q_1 = q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. If $q = q_1 > q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = ({\mathsf{r}_2}^{(>q_2)})^{(>q)} = ({\mathsf{r}_3}^{(>q_2)})^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Finally, if $q = q_2 > q_1$ then ${\mathsf{r}_1}^{(>q)} = ({\mathsf{r}_1}^{(>q_1)})^{(>q)} = ({\mathsf{r}_2}^{(>q_1)})^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$. Moreover, $q \in \mathsf{rng}\big({\mathsf{r}_2}^{(>q_1)}\big) = \mathsf{rng}\big({\mathsf{r}_1}^{(>q_1)}\big) \subseteq \mathsf{rng}(\mathsf{r}_1)$. Summing up, it holds that $s_1 \prec_\mathfrak{D} s_3$.

- Item 3.a for $s_1 \prec_\mathfrak{D} s_2$ and Item 3.b for $s_2 \prec_\mathfrak{D} s_3$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_1)$ with $q \geq p_1$ such that ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)}$ and $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$; moreover, $\mathsf{r}_2 = \mathsf{r}_3$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Consequently, $s_1 \prec_\mathfrak{D} s_3$.

- Item 3.a for $s_2 \prec_\mathfrak{D} s_3$ and Item 3.b for $s_1 \prec_\mathfrak{D} s_2$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_2)$ with $q \geq p_2$ such that ${\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q)$; moreover, $\mathsf{r}_1 = \mathsf{r}_2$ and $p_1 < p_2$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$, $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$, $q \in \mathsf{rng}(\mathsf{r}_1)$, and $q > p_1$. Consequently, $s_1 \prec_\mathfrak{D} s_3$.

- Item 3.b for both $s_1 \prec_\mathfrak{D} s_2$ and $s_2 \prec_\mathfrak{D} s_3$: $\mathsf{r}_1 = \mathsf{r}_2$, $\mathsf{r}_2 = \mathsf{r}_3$, $p_1 < p_2$, and $p_2 < p_3$. Hence, $\mathsf{r}_1 = \mathsf{r}_3$ and $p_1 < p_3$, which implies that $s_1 \prec_\mathfrak{D} s_3$ also in this case.

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

Lemma 6.3.2 (Query Function)**.** *The function $\Re_\mathfrak{D}$ is a query function,* i.e.*, for all states $s \in \mathrm{S}_\mathfrak{D}$, it holds that* (1) $\Re_\mathfrak{D}(s) \in \mathrm{QD}_\mathfrak{D}$ *and* (2) *if $\Re_\mathfrak{D}(s) \in \mathrm{QD}_\mathfrak{D}^-$ then $s \succ_\mathfrak{D} \Re_\mathfrak{D}(s)$.*

Proof. Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_\mathfrak{D}$ be a state and $(\mathrm{R}, \wp) \triangleq \Re_\mathfrak{D}(s)$ the pair of a set of positions $\mathrm{R} \subseteq \mathrm{Ps}_\mathfrak{D}$ and a player $\wp \in \{0, 1\}$ obtained by computing the function $\Re_\mathfrak{D}$ on s. Due to Line 1 of Algorithm 7, it follows that $\wp \equiv_2 p$. By Item 1.b, it holds that $p \in \mathsf{rng}(\mathsf{r})$. Thus, by Item 1.a and the definition of region function,

we have that $\mathsf{r}^{-1}(p)$ is an $\wp$-region in $\partial_{\mathsf{r}}^{\leq p}$, the latter being equal to ∂_s. Now, by Line 2 of Algorithm 7 and Proposition 6.3.2, where we assume $\mathrm{R}^\star \triangleq \mathsf{r}^{-1}(p)$ and $\partial \triangleq \partial_s$, it follows that $\mathrm{R} \supseteq \mathsf{r}^{-1}(p)$ is an $\wp$-region in ∂_s, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\partial_s}$, and, so a quasi dominion in ∂, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{QD}_{\partial}$. In addition, R is $\wp$-maximal in ∂_s. Consequently, $s \succ_{\partial} (\mathrm{R}, \wp)$, since all requirements of Definition 6.3.3 are satisfied. □

Lemma 6.3.3 (Successor Function)**.** *The function $\downarrow_{\partial}$ is an successor function,* i.e.*, for all states $s \in \mathrm{S}_{\partial}$ and region pairs $(\mathrm{R}, \wp) \in \mathrm{Rg}^{-}_{\partial}$ with $s \succ_{\partial} (\mathrm{R}, \wp)$, it holds that* (1) $s{\downarrow_{\partial}}(\mathrm{R}, \wp) \in \mathrm{S}_{\partial}$ *and* (2) $s{\downarrow_{\partial}}(\mathrm{R}, \wp) \prec_{\partial} s$.

Proof. Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}_{\partial}$ be a state, $(\mathrm{R}, \wp) \in \mathrm{Rg}^{-}_{\partial}$ an open region pair in ∂ compatible with s, and $s^\star = (\mathsf{r}^\star, p^\star) \triangleq s{\downarrow_{\partial}}(\mathrm{R}, \wp)$ the result obtained by computing the function $\downarrow_{\partial}$ on s and $(\mathrm{R}, \wp)$. Due to Item 1 of Definition 6.3.2, we have that *(1)* r is maximal above p and *(2)* $\mathsf{r}^{(<p)} \subseteq \mathsf{pr}_{\partial}{}^{(<p)}$. Moreover, by Item 1 of Definition 6.3.3, it holds that $\mathrm{R} \subseteq \mathrm{Ps}_{\partial_s}$, which implies *(3)* $\mathsf{dom}\big(\mathsf{r}^{(>p)}\big) \cap \mathrm{R} = \emptyset$.

On the one hand, suppose that R is $\wp$-open in ∂_s, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{Rg}^{-}_{\partial_s}$. By Lines 1-3 of Algorithm 8, we have that *(4)* $\mathsf{r}^\star = \mathsf{r}[\mathrm{R} \mapsto p]$ and *(5)* $p^\star = \mathsf{max}(\mathsf{rng}\Big(\mathsf{r}^{\star(<p)}\Big))$. In addition, by Item 2 of Definition 6.3.3, it holds that *(6)* R is $\wp$-maximal in ∂_s and *(7)* $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$. Now, by Point *(5)*, it immediately follows that *(8)* $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$, *(9)* $p^\star < p$, and *(10)* there is no priority $q \in \mathsf{rng}(\mathsf{r}^\star)$ such that $p^\star < q < p$. Also, by Points *(4)*, *(3)* and *(7)*, we have that $\mathsf{r}^{\star(>p)} = \mathsf{r}^{(>p)}$, $\mathsf{r}^{\star -1}(p) = \mathrm{R}$, and $\mathsf{r}^{\star(<p)} \subseteq \mathsf{r}^{(<p)}$. Thus, by Points *(1)*, *(6)*, *(9)*, and *(10)*, it holds that *(11)* $\mathsf{r}^\star$ is maximal above $p^\star$. Moreover, by Points *(2)* and *(9)*, we derive that *(12)* $\mathsf{r}^{\star(<p^\star)} \subseteq \mathsf{pr}_{\partial}{}^{(<p^\star)}$. Finally, to prove that *(13)* $\mathsf{r}^\star$ is a region function, we need to show that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{r}^\star}^{\leq q}}$, if non-empty, is a β-region in the subgame $\partial_{\mathsf{r}^\star}^{\leq q}$, for all priorities $q \in \mathsf{rng}(\mathsf{r}^\star)$ with $\beta \triangleq q \bmod 2$. Indeed, if $q > p$, by Point *(1)*, it holds that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{r}^\star}^{\leq q}} = \mathsf{r}^{-1}(q) \neq \emptyset$. Hence, the property follows directly from the fact that r is a region function. If $q = p$, again by Point *(1)*, we have that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{r}^\star}^{\leq q}} = \mathrm{R} \neq \emptyset$. So, the property is ensured by the hypothesis that R is an $\wp$-region. For the last case $q < p$, the property follows by Point *(12)* and the observation that, if non-empty, the set $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{r}^\star}^{\leq q}} \subseteq \mathsf{pr}_{\partial}^{-1}(q)$ only contains positions of priority q that necessarily form a region of the corresponding parity. Summing up, Points *(13)*, *(11)*, *(8)*, and *(12)* ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}_{\partial}$. At this point, it remains just to show that $(\mathsf{r}^\star, p^\star) \prec_{\partial} (\mathsf{r}, p)$. By Point *(7)*, two cases may arise. If $\mathsf{r}^{-1}(p) \subset \mathrm{R}$, the thesis follows from Item 3.a of Definition 6.3.2, where the priority q is set to p. On the contrary, if $\mathrm{R} = \mathsf{r}^{-1}(p)$, by Point *(4)*, we have that $\mathsf{r}^\star = \mathsf{r}$. Therefore, due to Point *(9)*, the thesis is derived from Item 3.b of the same definition.

On the other hand, suppose that R is $\wp$-closed in ∂_s, *i.e.*, $(\mathrm{R}, \wp) \notin \mathrm{Rg}^{-}_{\partial_s}$. By Lines 1, 4, and 5 of Algorithm 8, we have that *(14)* $p^\star = \mathsf{bep}^{\overline{\wp}}_{\partial}(\mathrm{R}, \mathsf{r})$ and *(15)* $\mathsf{r}^\star = \mathsf{pr}_{\partial} \uplus \mathsf{r}^{(\geq p^\star)}[\mathrm{R} \mapsto p^\star]$. It is not hard to see that, by Points *(14)* and *(7)*, the definition of bep and the fact that R is closed, we have that *(16)* $p^\star > p$. Now, by Points *(15)* and *(3)*, it follows that $\mathsf{r}^{\star(>p^\star)} = \mathsf{r}^{(>p^\star)}$, $\mathsf{r}^{\star -1}(p^\star) = \mathsf{r}^{-1}(p^\star) \cup \mathrm{R}$, and *(18)*

$\mathsf{r}^{\star(<p^\star)} \subseteq \mathsf{pr}_\partial{}^{(<p^\star)}$. Consequently, *(19)* $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$. Moreover, due to Points *(1)* and *(16)*, we have that *(20)* $\mathsf{r}^\star$ is maximal above $p^\star$. Finally, the proof that *(21)* $\mathsf{r}^\star$ is a region function easily follows by observing that $\mathsf{r}^{-1}(p^\star) \cup \mathrm{R}$ is an $\wp$-region, due to Proposition 6.3.1. Summing up, Points *(21)*, *(20)*, *(19)*, and *(18)* ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}_\partial$. At this point, as for the previous case, it remains to show that $(\mathsf{r}^\star, p^\star) \prec_\partial (\mathsf{r}, p)$. This fact easily follows from Item 3.a of Definition 6.3.2, where the priority q is set to $p^\star$, since, by Points *(3)* and *(16)*, we have that $\mathsf{r}^{-1}(p^\star) \cap \mathrm{R} = \emptyset$, so $\mathsf{r}^{-1}(p^\star) \subset \mathsf{r}^{-1}(p^\star) \cup \mathrm{R} = \mathsf{r}^{\star-1}(p^\star)$. □

6.3.2 Complexity of PP Dominion Space

To assess the complexity of the PP approach, we produce an estimate of the size and depth of the dominion space $\mathcal{R}_\partial$. This provides upper bounds for both the time and space complexities needed by the search procedure $\mathsf{src}_{\mathcal{R}_\partial}$ that computes dominions. By looking at the definition of state space $\mathcal{S}_\partial$, it is immediate to see that, for a game ∂ with n positions and k priorities, the number of states is bounded by k^{n+1}. Indeed, there are at most k^n functions $\mathsf{r} : \mathrm{Ps}_\partial \to \mathrm{Pr}_\partial$ from positions to priorities that can be used as region function of a state. Moreover, every such function can be associated with at most k current priorities.

Measuring the depth is a little trickier. A relatively coarse bound can be obtained by observing that there is an homomorphism from $\mathcal{S}_\partial$ to the well-founded partial order, in which the region function r of a state is replaced by a partial function $\mathsf{f} : \mathrm{Pr}_\partial \rightharpoonup [1, n]$ with the following properties: it assigns to each priority $p \in \mathsf{rng}(\mathsf{r})$ the size $\mathsf{f}(p)$ of the associated region $\mathsf{r}^{-1}(p)$. The order $(\mathsf{f}_1, p_1) \prec (\mathsf{f}_2, p_2)$ between two pairs is derived from the one on the states, by replacing $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$ with $\mathsf{f}_2(q) < \mathsf{f}_1(q)$. This homomorphism ensures that every chain in $\mathcal{S}_\partial$ corresponds to a chain in the new partial order. Moreover, there are exactly $\binom{n+k}{k}$ partial functions f such that $\sum_{p \in \mathsf{dom}(\mathsf{f})} \mathsf{f}(p) \leq n$. Consequently, thanks to Stirling's approximation of the factorial function, every chain cannot be longer than $k\binom{n+k}{k} = (n+k)\binom{n+k-1}{k-1} \leq (n+k)\left(\mathsf{e}(n/(k-1)+1)\right)^{k-1}$, where e is Euler's number.

Theorem 6.3.2 (Size & Depth Upper Bounds)**.** *The size and depth of a* PP *dominion space* $\mathcal{R}$ *with* $n \in \mathbb{N}_+$ *positions and* $k \in [1, n]$ *priorities are bounded by* k^{n+1} *and* $(n+k)\left(\mathsf{e}(n/(k-1)+1)\right)^{k-1} = \mathrm{O}\left(kn\left(\frac{\mathsf{e}n}{k-1}\right)^{k-1}\right)$*, respectively.*

Unfortunately, due to the reset operations performed after each promotion, an exponential worst-case can actually be built. Indeed, consider the game $\partial_{l,h}$ having all positions ruled by player 0 and containing h chains of length $2l+1$ that converge into a single position of priority 0 with a self loop. The i-th chain has a head of priority $2(2h-i)+1$ and a body composed of l blocks of two positions with priority $2i-1$ and $2i$, respectively. The first position in each block also has a self loop. An instance of this game

with $l = 2$ and $h = 4$ is depicted in Figure 6.3. The labels of the positions, deprived of the possible apexes, correspond to the associated priorities and the highlighted area at the bottom of the figure groups together the last blocks of the chains. Intuitively, the execution depth of the PP dominion space for this game is exponential, since the consecutive promotion operations performed on each chain can simulate the increments of a counter up to l. Also, the priorities are chosen in such a way that, when the i-th counter is incremented, all the j-th counters with $j \in]i, h]$ are reset. Therefore, the whole game simulates a counter with h digits taking values from 0 to l. Hence, the overall number of performed promotions is $(l+1)^h$. The search procedure on $\mathfrak{D}_{2,4}$ starts by building the four open 1-regions $\{15\}$, $\{13\}$, $\{11\}$, and $\{9\}$ and the open 0-region $\{8, 7', 8'\}$, where we use apexes to distinguish different positions with the same priority. This state represents the configuration of the counter, where all four digits are set to 0. The closed 1-region $\{7\}$ is then found and promoted to 9. Consequently, the previously computed 0-region with priority 8 is reset and the new region is maximized to obtain the open 1-region $\{9, 7, 8\}$. Now, the counter is set to 0001. After that, the open 0-region $\{8'\}$ and the closed 1-region $\{7'\}$ are computed. The latter one is promoted to 9 and maximized to attract position $8'$. This completes the 1-region containing the entire chain ending in 9. The value of the counter is now 0002. At this point, immediately after the construction of the open 0-region $\{6, 5', 6'\}$, the closed 1-region $\{5\}$ is found, promoted to 11, and maximized to absorb position 6. Due to the promotion, the positions in the 1-region with priority 9 are reset to their original priority and all the work done to build it gets lost. This last operation represents the reset of the least significant digit of the counter, caused by the increment of the second one, *i.e.*, the counter displays 0010. Following similar steps, the process carries on until each chain is grouped in a single region. The corresponding state represents the configuration of the counter in which all digits are set to l. Thus, after an exponential number promotions, the closed 0-region $\{0\}$ is eventually obtained as solution.

Theorem 6.3.3 (Execution-Depth Lower Bounds)**.** *For all numbers $h \in \mathbb{N}$, there exists a* PP *dominion space $\mathcal{D}_h^{\mathrm{PP}}$ with $k = 2h + 1$ positions and priorities, whose execution depth is $3 \cdot 2^h - 2 = \Theta\big(2^{k/2}\big)$. Moreover, for all numbers $l \in \mathbb{N}_+$, there exists a* PP *dominion space $\mathcal{D}_{l,h}^{\mathrm{PP}}$ with $n = (2l+1) \cdot h + 1$ positions and $k = 3h + 1$ priorities, whose execution depth is $((3l+1) \cdot (l+1)^h - 1)/l - 2 = \mathrm{O}\Big((3n/(2(k-1)))^{k/3}\Big)$.*

Proof. The single-player game $\mathfrak{D}_h^{\mathrm{PP}}$, with $k = 2h + 1$ positions and priorities, contains a position with priority 0 plus $2h$ positions spanning all the odd priorities from 1 to $4h - 1$. Moreover, the associated moves are those depicted in the highlighted top section of Figure 6.3. Intuitively, this game encodes a binary counter, where each one of the h chains, composed of two positions of odd priorities connected to position 0, represents a digit. The promotion operation of region $2i-1$ to region $2(2h-i)+1$ corresponds

to the increment of the i-th digit. As a consequence, the number of promotions for the PP algorithm is equal to the number of configurations of the counter, except for the initial one. Now, to provide a lower bound for the depth of the associated dominion space $\mathcal{D}_h^{\mathrm{PP}}$, consider the recursive function $\mathsf{Q}(h)$ with base case $\mathsf{Q}(0) = 1$ and inductive case $\mathsf{Q}(h) = 2\mathsf{Q}(h-1) + 2$. This function counts the number of queries executed by the the PP algorithm on game $\mathfrak{I}_h^{\mathrm{PP}}$. The correctness of the base case is trivial. Indeed, when h is equal to 0, there are no chains and the only possible region is the one containing position 0. Let us now consider the inductive case $h > 0$. A first query is required to obtain region $\{4h-1\}$. Then, before reaching the promotion of region $\{1\}$ to region $\{4h-1\}$, *i.e.*, to set the most-significant digit, all the other digits must be set. To do this, the number of necessary queries is equal to those required on the game instance with $h-1$ chains minus the query that computes region $\{0\}$, *i.e.*, $\mathsf{Q}(h-1)-1$. At this point, two additional queries are counted, when region $\{1\}$ is obtained and then merged to region $\{4h-1\}$. As described above in the execution of the game depicted in Figure 6.3, this promotion unnecessarily resets all lesser-significant digits. Therefore, $\mathsf{Q}(h-1)-1$ counts exactly the queries needed to set again to 1 all the digits reset after this promotion. Finally, a query for the 0-closed region $\{0\}$ is required. Summing up, we have $\mathsf{Q}(h) = 1 + (\mathsf{Q}(h-1)-1) + 2 + (\mathsf{Q}(h-1)-1) + 1 = 2\mathsf{Q}(h-1) + 2$, as claimed above. A trivial proof by induction shows that $\mathsf{Q}(h) = 3 \cdot 2^h - 2 = \Theta\big(2^{k/2}\big)$.

We can now turn to the more complex single-player game $\mathfrak{I}_{l,h}^{\mathrm{PP}}$, with $n = (2l+1) \cdot h + 1$ positions and $k = 3h + 1$ priorities, which contains a position with priority 0 plus $(2l+1) \cdot h$ positions, spanning all the priorities from 1 to $2h$ and all odd priorities from $2h+1$ to $4h-1$ as depicted in the entire Figure 6.3. To provide a lower bound for the depth of the associated dominion space $\mathcal{D}_{l,h}^{\mathrm{PP}}$, consider the recursive function $\mathsf{Q}(h)$ with base case $\mathsf{Q}(l,0) = 1$ and inductive case $\mathsf{Q}(l,h) = (1+l)\mathsf{Q}(l,h-1) + 2l + 1$. Also in this case, the correctness of the base cases is trivial. Indeed, when h is equal to 0, there are no chains, independently of the parameter l. Hence, the only possible region is the one containing position 0. For the inductive case $h > 0$, a first query is required to compute region $\{4h-1\}$. Then, before reaching the promotion to region $\{4h-1\}$ of the adjacent region $\{1\}$, *i.e.*, to set the most-significant digit to 1, all the other digits must be set to $l-1$. This requires the same number of queries required to process the game with $h-1$ chains minus the query collecting region $\{0\}$, *i.e.*, $\mathsf{Q}(l,h-1)-1$. At this point, three additional queries are counted: region $\{2, 1', 2', \ldots\}$ is computed first, followed by region $\{1\}$; the latter is then merged to region $\{4h-1\}$ absorbing position 2 from the first region. Note that the region of priority 2 contains all positions in the first chain, except for positions 1 and $4h-1$. As in the previous game, such a promotion unnecessarily resets all lesser-significant digits. Therefore, $\mathsf{Q}(l,h-1)-1$ counts exactly the queries needed to set again to $l-1$ all digits reset after this promotion. After that, and similarly to what is described above, we need to set the most-significant

digit to 2, by performing other three queries: $\{2', \ldots\}$ is computed first, followed by region $\{1'\}$; the latter is then merged to region $\{4h-1, 1, 2\}$, absorbing position $2'$ from the first region. Again, this promotion resets all lesser-significant digits, which requires $\mathsf{Q}(l, h-1)-1$ steps to be set to $l-1$. This behavior is repeated $l-2$ more times, until the first chain is completely absorbed by the region of highest priority $4h-1$. Finally, a query for the 0-closed region $\{0\}$ is required. Summing up, we have $\mathsf{Q}(n,h) = 1+(\mathsf{Q}(h-1)-1)+(3+(\mathsf{Q}(l,h-1)-1))\cdot l+1 = (1+l)\mathsf{Q}(l,h-1)+2l+1$, as claimed above. Also in this case, an easy induction shows that $\mathsf{Q}(l,h) = ((3l+1)\cdot(l+1)^h-1)/l-2 = \mathrm{O}\Big((3n/(2(k-1)))^{k/3}\Big)$. □

Observe that, in the above theorem, we provide two different exponential lower bounds. The general one, with $k/3$ as exponent and a parametric base, is the result of the game $\partial_{l,h}$ described in the previous paragraph, where $k = 3h+1$. The other bound, instead, has a base fixed to 2, but a larger exponent $k/2$. We conjecture that the given upper bound could be improved to match the exponent $k/2$ of this lower bound. In this way, we would obtain an algorithm with an asymptotic behavior comparable with the one exhibited by the small-progress measure procedure [Jur00].

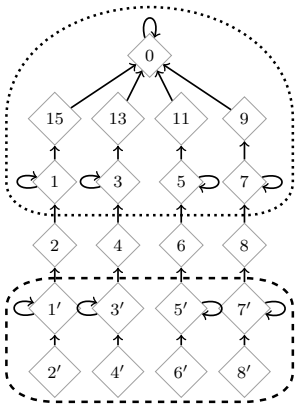


Figure 6.3: The $\partial^{\mathrm{PP}}_{2,4}$ game.

6.4 Subgame Decomposition

As described in the previous section, the search procedure for the basic PP approach, displayed in Algorithm 6, consists in finding quasi dominions that, whenever closed in the current subgame, are suitably merged with previously computed regions until a dominion in the original game ∂ is found. In particular, when the current quasi dominion is open, the algorithm searches for the next one in a subgame ∂' of ∂. By Algorithm 7 and Lines 2-3 of Algorithm 8, such a subgame corresponds to $\partial_s \setminus \mathrm{R}$, where ∂_s and R are, respectively, the current subgame *w.r.t.* to state s and the region extracted by the query function on s itself. Due to this design choice and the fact that the regions are created in decreasing order of priority, the sequence of subgames follows the same order and, in particular, the current priority p contained in the state $s = (\mathsf{r}, p)$ also corresponds to the maximal priority in the part of game yet to be analyzed. This is a very simple and efficient way to compute subgames, but it does not necessarily translate into an efficient way to obtain a solution in terms of the number of promotions required. In fact, the lesser the moves between ∂' and the positions outside ∂', the easier it is to find a dominion in the subsequent iterations, as the probability that a promotion occurs reduces. In this section

we propose a generalization of the PP algorithm that does not commit to a specific way of computing the next subgame to process during its dominion search phase. The result is a parametric version of the priority promotion approach that, by exploiting the topology of the underlying game, can be instantiated with different game decomposition techniques.

To this end, we first need to address the crucial issue that the correctness of promotion mechanism relies, according to Proposition 6.3.1, on the fact that the positions contained in a dominion to be promoted can only reach some previously computer region. To ensure that this property is preserved, we need to require that, after computing region R in game ∂_s, whatever the subgame ∂' of $\partial_s \setminus \mathrm{R}$ we chose to consider next, it cannot contain positions that reach portions of the game yet to be analyzed, namely positions in the set $\mathrm{Ps}_{\partial_s} \setminus (\mathrm{R} \cup \mathrm{Ps}_{\partial'})$. Since there is no unique way to choose a suitable subgame, the new algorithm we propose in the following is parametric on a function F that, given a game ∂, computes the positions of the subgame ∂' on which the search for quasi dominions needs to proceed. The standard PP algorithm can be obtained by simply setting $\mathsf{F}(\partial) = \mathrm{Ps}_{\partial}$. A sharper heuristic, instead, instantiates F with a procedure that collects the positions in some minimal Strongly Connected Component (SCC) of the graph underlying the game ∂. An alternative coarser possibility, which, however, incurs in less overhead, simply computes all the positions reachable from an arbitrary initial position in the game. It is immediate to see that all these instantiations satisfy the above requirement on the subgames.

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

A final issue involves the way we keep track of the computed regions. In the PP procedure, we use region function r, which is initialized to $\mathsf{pr}_{\mathcal{D}}$ and directly stores the stratification of the regions computed in decreasing order of priority. Therefore, for every priority p, the set $\mathsf{r}^{-1}(p)$ exactly corresponds to a region R in $\mathcal{D}_s = \mathcal{D}^{\leq p}_{\mathsf{r}}$, where $s \triangleq (\mathsf{r}, p)$. If the subgame $\mathcal{D}_s$ is computed via the subgame function g, instead, a problem with the original definition of the region function r arises: r might contain positions associated with priorities that may not exist in g. Note that this problem is a direct consequence of the strict inclusion $\mathcal{D}' \subset \mathcal{D}_s \setminus \mathrm{R}$. For this reason, we generalize the notion of region function r to a *quasi-dominion function* q. The regions computed so far can still be recovered by restricting q to the current subgame, identified by g, in symbols $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. Formally, given a game $\mathcal{D}$, a priority function $\mathsf{q} \in \mathbb{Q}_{\mathcal{D}} \subseteq \mathrm{Ps}_{\mathcal{D}} \to \mathrm{Pr}_{\mathcal{D}}$ is a *quasi-dominion function* iff, for all $p \in \mathsf{rng}(\mathsf{q})$, it holds that $\mathsf{q}^{-1}(p)$ is an $\wp$-quasi dominion in $\mathcal{D}$, with $\wp \triangleq p \bmod 2$. We say that the function $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$ is a g-*stratified region function* if *(i)* $\mathsf{r}^{-1}(p)$ is an $\wp$-region in $\mathcal{D}^{\leq p}_{\mathsf{g}}$, for all $p \in \mathsf{rng}(\mathsf{r})$, and *(ii)* $\mathsf{esc}^{\overline{\wp}}_{\mathcal{D}}(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\wp}}_{\mathcal{D}}(\mathrm{X}) = \emptyset$, for each $p \in \mathsf{rng}(\mathsf{g})$ and quasi dominion pair $(\mathrm{Q}, \wp) \in \mathrm{QD}_{\mathcal{D}^{\leq p}_{\mathsf{g}}}$, where $\mathrm{X} = \mathrm{Ps}_{\mathcal{D}} \setminus (\mathsf{dom}(\mathsf{g}^{(\leq p)}) \cup \mathsf{dom}(\mathsf{r}))$. Moreover, r is *maximal* w.r.t. g if $\mathsf{r}^{-1}(p)$ is $\wp$-maximal in $\mathcal{D}^{\leq p}_{\mathsf{g}}$, for all $p \in \mathsf{rng}(\mathsf{r})$ with $p > \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ and $\wp = p \bmod 2$. Intuitively, Items *(i)* of the g-stratified property and the maximality of r *w.r.t.* g state the same two requirements that a classic PP region function must satisfy. In addition, Item *(ii)* of the former property precisely formalizes the above described requirement on the quasi dominions: every quasi dominion in a subgame identified by the function g cannot have positions with moves that lead outside the domain of the function r.

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

Definition 6.4.1 (State Space)**.** *A* state space *is a tuple* $\mathcal{S}_{\mathcal{D}} \triangleq \langle \mathrm{S}_{\mathcal{D}}, \top_{\mathcal{D}}, \prec_{\mathcal{D}} \rangle$*, where its components are defined as prescribed in the following:*

1. $\mathrm{S}_{\mathcal{D}} \subseteq \mathbb{Q}_{\mathcal{D}} \times \mathbb{G}_{\mathcal{D}}$ *is the set of all pairs* $s \triangleq (\mathsf{q}, \mathsf{g})$*, called* states*, composed of a quasi dominion function* $\mathsf{q} \in \mathbb{Q}_{\mathcal{D}}$ *and a subgame function* $\mathsf{g} \in \mathbb{G}_{\mathcal{D}}$ *such that* (a) $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$ *is maximal* w.r.t. g, (b) r *is a* g*-stratified region function, and* (c) $\mathsf{q} = \mathsf{pr}_{\mathcal{D}} \uplus \mathsf{r}$*;*

2. $\top_{\mathcal{D}} \triangleq (\mathsf{pr}_{\mathcal{D}}, \varnothing[\mathrm{Ps}_{\mathcal{D}} \mapsto \mathsf{pr}(\mathcal{D})])$*;*

3. *for any two states* $s_1 \triangleq (\mathsf{q}_1, \mathsf{g}_1), s_2 \triangleq (\mathsf{q}_2, \mathsf{g}_2) \in \mathrm{S}_{\mathcal{D}}$*, with* $p_1 \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}_1))$ *and* $p_2 \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}_2))$*, it holds that* $s_1 \prec_{\mathcal{D}} s_2$ *iff either* (a) *there exists a priority* $p \in \mathsf{rng}(\mathsf{q}_1)$ *with* $p \geq p_1$ *such that* (a.i) $\mathsf{q}_1{}^{(>p)} = \mathsf{q}_2{}^{(>p)}$ *and* (a.ii) $\mathsf{q}_2^{-1}(p) \subset \mathsf{q}_1^{-1}(p)$ *or* (b) *both* (b.i) $\mathsf{q}_1 = \mathsf{q}_2$ *and* (b.ii) $p_1 < p_2$ *hold.*

As one would expect, the state space defined above is slightly different from the one in Section 6.3. Indeed, a state (q, g) does not explicitly record the regions anymore. Instead, it records the quasi dominions from which the computed regions can be extracted. In more detail, an $\wp$-maximal region is obtained from the derived set $\mathsf{r}^{-1}(q) = \mathsf{q}^{-1}(q) \cap \mathsf{g}^{-1}(q)$, where $q \geq \mathsf{min}(\mathsf{rng}(\mathsf{g}))$, $\wp = q \bmod 2$, and $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. Obviously, the current priority $p \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ of the state must be a priority of an actual region in r. Moreover, the quasi dominion function q, at any priority p' lower than p, must contain positions with the same priority p' in the original priority function $\mathsf{pr}_{\mathcal{D}}$, *i.e.*, $\mathsf{q}^{-1}(p') \subseteq \mathsf{pr}_{\mathcal{D}}^{-1}(p')$. At any p' grater that p, instead, q needs to store part of the priority function together with the regions, that is $\mathsf{q}^{-1}(p') = (\mathsf{pr}_{\mathcal{D}} \uplus \mathsf{r})^{-1}(p')$. All these requirement are ensured by Points *(a)-(c)* of Item 1. Finally, the order follows exactly the same rules as in the original algorithm, where the pair $(\mathsf{q} \cap \mathsf{g}, \mathsf{min}(\mathsf{rng}(\mathsf{g})))$ plays the role of the PP state (r, p).

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

Definition 6.4.2 (Compatibility Relation)**.** *An open quasi dominion pair* $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}_{\mathcal{D}}$ *is* compatible *with a state* $s \triangleq (\mathsf{q}, \mathsf{g}) \in \mathrm{S}_{\mathcal{D}}$*, in symbols* $s \succ_{\mathcal{D}} (\mathrm{R}, \wp)$*, iff* (1) $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\mathcal{D}_s}$ *and* (2) *if* R *is* $\wp$*-open in* $\mathcal{D}_s$ *then* (2.a) R *is* $\wp$*-maximal in* $\mathcal{D}_s$ *and* (2.b) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$*, where* $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$.

Algorithm 9 provides the implementation for the query function compatible with the generalized priority-promotion mechanism. Let $s \triangleq (\mathsf{q}, \mathsf{g})$ be the current state. Line 1 extracts from s the priority p to process, while Line 2 simply computes the associated parity $\wp$. Finally, Line 3 computes the attractor *w.r.t.* player $\wp$ of the quasi dominion contained in q at p that belongs to the subgame $\mathcal{D}_s$, *i.e.*, the $\wp$-region $\mathsf{q}^{-1}(p) \cap \mathrm{Ps}_{\mathcal{D}_s} = \mathsf{q}^{-1}(p) \cap \mathsf{g}^{-1}(p) = \mathsf{r}^{-1}(p)$. The resulting set R is, according to Proposition 6.3.2, an $\wp$-maximal $\wp$-region in $\mathcal{D}_s$ that contains $\mathsf{r}^{-1}(p)$.

Algorithm 9: New Query Function.

```
   signature ℜ_D : S_D → (2^{Ps_D} × {0,1})
   function ℜ_D(s)
       let (q, g) = s in
1          p ← min(rng(g))
2          ℘ ← p mod 2
3          R ← atr^℘_{D_s}(q^{-1}(p) ∩ Ps_{D_s})
4      return (R, ℘)
```

Unlike for the classic PP algorithm, the notion of *best escape priority* for player $\overline{\wp}$ *w.r.t.* an $\wp$-region R of the subgame $\mathcal{D}_s$ is defined *w.r.t.* the quasi dominion function q. Indeed, we do not need to consider the g-stratified region function r, since, due to the property of the subgame function g, the escape positions in every quasi dominion in $\mathcal{D}_s$ can only reach positions in r. This means that the range of the interface relation I is included in r itself. At this point, the successor function can be defined

parametrically on the function F that identifies a subgame $\partial' \subseteq \partial \setminus \mathrm{R}$ such that, for each player $\wp$ and quasi $\wp$-dominion $\mathrm{Q} \in \mathrm{QD}_{\partial'}$, the set $\mathsf{esc}^{\overline{\wp}}_{\partial'}(\mathrm{Q})$ has no positions that reach $\partial \setminus (\mathrm{R} \cup \mathrm{Ps}_{\partial'})$. More formally, the set $\mathrm{A} \triangleq \mathsf{F}(\partial) \subseteq \mathrm{Ps}_{\partial}$ must be such that the subgame $\partial{\restriction}\mathrm{A}$ satisfies $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\wp}}_{\partial}(\mathrm{Ps}_{\partial} \setminus \mathrm{A}) = \emptyset$, for every quasi dominion pair $(\mathrm{Q}, \wp) \in \mathrm{QD}_{\partial{\restriction}\mathrm{A}}$. Observe that this requirement exactly mirrors Item *(ii)* in the definition of the g-stratified region function r. As already said before, we propose two instances of such a function. The first one computes the reachability set of an arbitrary position. The second one, instead, simply returns the positions of a minimal SCC. Clearly the escape of every quasi dominion in both the above subgames cannot reach positions outside those subgames.

Algorithm 10 implements the new successor function. Given the current state s and a compatible region pair $(\mathrm{R}, \wp)$ open in the whole input game, it produces a successor state $s^\star \triangleq (\mathsf{q}^\star, \mathsf{g}^\star)$ in the dominion space. In particular, it first checks whether R is open also in the subgame ∂_s (Line 1). If this is the case, it assigns priority $p = \mathsf{min}(\mathsf{rng}(\mathsf{g}))$ to region R and stores it in the new quasi dominion function $\mathsf{q}^\star$ (Line 2). The set A of positions of the new subgame is computed via the auxiliary function F (Line 3) and stored in the subgame function $\mathsf{g}^\star$ at priority $\mathsf{max}(\mathsf{rng}(\mathsf{q}^\star \restriction \mathrm{A}))$ (Line 4). If, on the other hand, R is closed in ∂_s, a promotion is performed. The next priority $p^\star$ is set to the bep of R *w.r.t.* q for player $\overline{\wp}$ in the entire game ∂ (Line 5). Region R is, then, promoted to priority $p^\star$ and all the positions with priorities smaller than $p^\star$ in the current quasi dominion function q are reset (Line 6). Similarly, in the new subgame function $\mathsf{g}^\star$, the priorities of the same positions are set to $p^\star$ (Line 7). Also in this case, the correctness of this last operation follows from Proposition 6.3.1.

Algorithm 10: New Successor Function.

```
   signature ↓ : ≻ → Δ × Pr
   function s↓(R, ℘)
       let (q, g) = s in
1          if (R, ℘) ∈ Rg⁻_{∂_s} then
2              q* ← q[R ↦ min(rng(g))]
3              A ← F(∂_s \ R)
4              g* ← g[A ↦ max(rng(q* ↾ A))]
           else
5              p* ← bep^{℘̄}_∂(R, q)
6              q* ← pr_∂ ⊎ q^{(≥p*)}[R ↦ p*]
7              g* ← g[dom(g^{(<p*)}) ↦ p*]
8          return (q*, g*)
```

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

Theorem 6.4.1 (Dominion Space)**.** *For a game ∂, the* structure $\mathcal{D}_\partial \triangleq \langle \partial, \mathcal{S}_\partial, \succ_\partial, \Re_\partial, \downarrow_\partial \rangle$*, where $\mathcal{S}_\partial$ is given in Definition 6.4.1, $\succ_\partial$ is the relation of Definition 6.4.2, and $\Re_\partial$ and $\downarrow_\partial$ are the functions*

computed by Algorithms 9 and 10 is a dominion space.

Similarly to the proof of Theorem 6.3.1, to prove the above theorem, we have to show that the three components $\mathcal{S}_{\mathfrak{D}}$, $\Re_{\mathfrak{D}}$, and $\downarrow_{\mathfrak{D}}$ of the structure $\mathcal{D}_{\mathfrak{D}}$ satisfy the properties required by Definition 6.4.1 of dominion space. We do this through the Lemmas 6.4.1, 6.4.2, and 6.4.3.

Lemma 6.4.1 (State Space)**.** *The generalized* PP *state space* $\mathcal{S}_{\mathfrak{D}} = \langle \mathrm{S}_{\mathfrak{D}}, \top_{\mathfrak{D}}, \prec_{\mathfrak{D}} \rangle$ *for a game* $\mathfrak{D} \in \mathcal{P}$ *is a well-founded partial order* w.r.t. $\prec_{\mathfrak{D}}$ *with designated element* $\top_{\mathfrak{D}} \in \mathrm{S}_{\mathfrak{D}}$.

Proof. To prove that $\prec_{\mathfrak{D}}$ is a strict partial order on $\mathrm{S}_{\mathfrak{D}}$, we can follow the same approach used in the proof of Theorem 6.3.1, where the pair $(\mathsf{q} \cap \mathsf{g}, \mathsf{min}(\mathsf{rng}(\mathsf{g})))$ plays the same role of the state (r, p) in the PP algorithm. Therefore, to complete the proof, we only need to show that $\top_{\mathfrak{D}} \triangleq (\mathsf{pr}_{\mathfrak{D}}, \varnothing[\mathrm{Ps}_{\mathfrak{D}} \mapsto \mathsf{pr}(\mathfrak{D})])$ belongs to $\mathrm{S}_{\mathfrak{D}}$. The function $\mathsf{q} \triangleq \mathsf{pr}_{\mathfrak{D}}$ is trivially a quasi dominion function, since $\mathsf{pr}_{\mathfrak{D}}^{-1}(p)$ is an $\wp$-quasi dominion in the original game $\mathfrak{D}$ as it only contains positions of the same priority p of parity $\wp$, for all $p \in \mathsf{rng}(\mathsf{pr}_{\mathfrak{D}})$ with $\wp \triangleq p \bmod 2$. To prove that $\mathsf{g} \triangleq \varnothing[\mathrm{Ps}_{\mathfrak{D}} \mapsto \mathsf{pr}(\mathfrak{D})]$ is a subgame function, first observe that Item *(i)* of the corresponding definition holds due to the fact that $\mathsf{rng}(\mathsf{g}) = \{\mathsf{pr}(\mathfrak{D})\}$ and $\mathsf{pr}(\mathfrak{D}) \in \mathsf{rng}(\mathsf{pr}_{\mathfrak{D}})$. Moreover, $\mathsf{dom}\big(\mathsf{g}^{(\leq \mathsf{pr}(\mathfrak{D}))}\big) = \mathrm{Ps}_{\mathfrak{D}}$, so, $\mathfrak{D}_{\mathsf{g}}^{\leq \mathsf{pr}(\mathfrak{D})} = \mathfrak{D}$ is clearly a game. Consequently, Item *(ii.a)* holds as well. Finally, Item *(ii.b)* is vacuously verified, since there is no priority $p \in \mathsf{rng}(\mathsf{g})$ with $p > \mathsf{pr}(\mathfrak{D})$. We can now consider the three points in Item 1 of Definition 6.4.1. Point *(a)* vacuously holds, since, as already observed, there is no priority in g greater than $\mathsf{pr}(\mathfrak{D})$. Now, let $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. First notice that $\mathsf{r}^{-1}(\mathsf{pr}(\mathfrak{D})) = \mathsf{pr}_{\mathfrak{D}}^{-1}(\mathsf{pr}(\mathfrak{D}))$ is a region in $\mathfrak{D}_{\top_{\mathfrak{D}}} = \mathfrak{D}_{\mathsf{g}}^{\leq \mathsf{pr}(\mathfrak{D})} = \mathfrak{D}$, since it is a quasi dominion whose escape positions have the maximal priority in the game. Moreover, $\mathrm{X} = \mathrm{Ps}_{\mathfrak{D}} \setminus (\mathsf{dom}\big(\mathsf{g}^{(\leq \mathsf{pr}(\mathfrak{D}))}\big) \cup \mathsf{dom}(\mathsf{r})) = \mathrm{Ps}_{\mathfrak{D}} \setminus \mathrm{Ps}_{\mathfrak{D}} = \emptyset$. Consequence, Point *(b)* also holds. Finally, Point *(c)* is implied by the fact that $\mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{r} = \mathsf{pr}_{\mathfrak{D}} \uplus (\mathsf{q} \cap \mathsf{g}) = \mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{q} = \mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{pr}_{\mathfrak{D}} = \mathsf{pr}_{\mathfrak{D}} = \mathsf{q}$. □

Lemma 6.4.2 (Query Function)**.** *The function* $\Re_{\mathfrak{D}}$ *is a query function,* i.e.*, for all states* $s \in \mathrm{S}_{\mathfrak{D}}$*, it holds that* (1) $\Re_{\mathfrak{D}}(s) \in \mathrm{QD}_{\mathfrak{D}}$ *and* (2) *if* $\Re_{\mathfrak{D}}(s) \in \mathrm{QD}_{\mathfrak{D}}^{-}$ *then* $s \succ_{\mathfrak{D}} \Re_{\mathfrak{D}}(s)$.

Proof. Let $s \triangleq (\mathsf{q}, \mathsf{g}) \in \mathrm{S}_{\mathfrak{D}}$ be a state and $(\mathrm{R}, \wp) \triangleq \Re_{\mathfrak{D}}(s)$ the pair of a set of positions $\mathrm{R} \subseteq \mathrm{Ps}_{\mathfrak{D}}$ and a player $\wp \in \{0, 1\}$ obtained by computing the function $\Re_{\mathfrak{D}}$ on s. Moreover, assume $\mathsf{r} \triangleq \mathsf{q} \cap \mathsf{g}$. First observe that Item 1.c implies $\mathsf{rng}(\mathsf{g}) \subseteq \mathsf{rng}(\mathsf{q})$. Indeed, suppose by contradiction that there exists a priority $p \in \mathsf{rng}(\mathsf{g}) \setminus \mathsf{rng}(\mathsf{q})$. Clearly, $p \notin \mathsf{rng}(\mathsf{r})$. Moreover, $p \in \mathsf{rng}(\mathsf{g}) \subseteq \mathsf{rng}(\mathsf{pr}_{\mathfrak{D}})$, due to Item *(i)* of the definition of subgame function applied to g. Therefore, $p \in \mathsf{rng}(\mathsf{pr}_{\mathfrak{D}} \uplus \mathsf{r}) = \mathsf{rng}(\mathsf{q})$, but this is obviously impossible. Now, by Line 1 of Algorithm 9 and the definition of the subgame $\mathfrak{D}_s$, we have that $\mathsf{q}^{-1}(p) \cap \mathrm{Ps}_{\mathfrak{D}_s} = \mathsf{q}^{-1}(p) \cap \mathsf{dom}\big(\mathsf{g}^{(\leq p)}\big) = \mathsf{q}^{-1}(p) \cap \mathsf{g}^{-1}(p) = \mathsf{r}^{-1}(p) \neq \emptyset$. Thus, due to Line 2 of the same algorithm and Item 1.b, it holds that $\mathsf{r}^{-1}(p)$ is an $\wp$-region in $\mathfrak{D}_s$. Finally, by Line 3 and Proposition 6.3.2,

R is $\wp$-maximal in ∂_s. Consequently, both Point *(1)* and the requirements of Definition 6.3.3 are satisfied. □

Lemma 6.4.3 (Successor Function)**.** *The function* $\downarrow_\partial$ *is an successor function,* i.e.*, for all states* $s \in \mathrm{S}_\partial$ *and quasi dominion pairs* $(\mathrm{R}, \wp) \in \mathrm{QD}^-_\partial$ *with* $s \succ_\partial (\mathrm{R}, \wp)$*, it holds that* (1) $s\downarrow_\partial (\mathrm{R}, \wp) \in \mathrm{S}_\partial$ *and* (2) $s\downarrow_\partial (\mathrm{R}, \wp) \prec_\partial s$*.*

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

In the first case, by Lines 1-4 of Algorithm 10, we have that *(1)* $\mathsf{q}^\star = \mathsf{q}[\mathrm{R} \mapsto p]$, *(2)* $\mathrm{A} = \mathsf{F}(\partial_s \setminus \mathrm{R})$, and *(3)* $\mathsf{g}^\star = \mathsf{g}[\mathrm{A} \mapsto \mathsf{max}(\mathsf{rng}(\mathsf{q}^\star \restriction \mathrm{A}))]$. By Definition 6.4.2, it holds that $\mathrm{R} \subseteq \mathrm{Ps}_{\partial_s}$. Thus, *(4)* $\mathrm{R} \subseteq \mathsf{g}^{-1}(p)$, *i.e.* R does not contain positions with priority higher than p. As consequence of Points *(1)* and *(4)*, we have that *(5)* $\mathsf{q}^{\star(>p)} = \mathsf{q}^{\star(>p)}$, *(6)* $\mathsf{q}^{\star(<p)} \subseteq \mathsf{q}^{(<p)}$, and *(7)* $\mathsf{q}^{\star -1}(p) = \mathrm{R} \cup \mathsf{q}^{-1}(p)$.

By Point *(5)* and the assumption on q, we have that $\mathsf{q}^{\star -1}(p')$ is an $\wp'$-quasi dominion in ∂, for all priorities $p \in \mathsf{rng}(q)$ with $p' > p$ and $\wp' \triangleq p' \bmod 2$. Moreover, by Item 1.c of Definition 6.4.1, for every priority $p \in \mathsf{rng}(q)$ with $p' < p$ and $\wp' \triangleq p' \bmod 2$, it holds that *(8)* $\mathsf{q}^{-1}(p') \subseteq \mathsf{pr}^{-1}_\partial(p')$, so, by Point *(6)*, $\mathsf{q}^{\star -1}(p')$ is an $\wp'$-quasi dominion in ∂ as well. Again by Item 1.c, we have that $\mathsf{q}^{-1}(p)$ is the union of $\mathsf{r}^{-1}(p)$ with a set $\mathrm{P} \subseteq \mathsf{pr}^{-1}_\partial(p) \setminus \mathsf{g}^{-1}(p)$. Moreover, *(9)* $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$, due to Definition 6.4.2. Thus, by Point *(7)*, it holds that *(10)* $\mathsf{q}^{\star -1}(p) = \mathrm{R} \cup \mathsf{q}^{-1}(p) = \mathrm{R} \cup \mathrm{P} \cup \mathsf{r}^{-1}(p) = \mathrm{R} \cup \mathrm{P}$, *i.e.*, $\mathsf{q}^{\star -1}(p)$ is the union of an $\wp$-quasi dominion in ∂ with some set of positions of priority p having the same parity. Consequently, also $\mathsf{q}^{\star -1}(p)$ is a $\wp$-quasi dominion in ∂. In conclusion, we have that *(11)* $\mathsf{q}^\star$ is a quasi-dominion function.

By Point *(2)* and the assumption on the function F, we know that *(12)* $\emptyset \neq \mathrm{A} \subseteq \partial_s \setminus \mathrm{R} = \mathsf{dom}\big(\mathsf{g}^{\leq p}\big) \setminus \mathrm{R}$ induces a subgame $\partial \restriction \mathrm{A}$ of $\partial_s \setminus \mathrm{R}$ such that $\mathsf{esc}^{\overline{\wp}}_\partial(\mathrm{Q}) \cap \mathsf{pre}^{\overline{\wp}}_\partial(\mathrm{Ps}_\partial \setminus \mathrm{A}) = \emptyset$, for every quasi dominion pair $(\mathrm{Q}, \wp) \in \mathrm{QD}_{\partial \restriction \mathrm{A}}$. In addition, it is easy to see that $\mathsf{pr}^{-1}_\partial(p) \cap \mathrm{Ps}_{\partial_s} \subseteq \mathrm{R}$. Indeed, if by contradiction this was not the case, by definition of the subgame ∂_s and Point *(9)*, there would be a position $v \in \mathsf{pr}^{-1}_\partial(p) \cap \mathsf{g}^{-1}(p)$ such that $v \notin \mathsf{r}^{-1}(p)$. Therefore, due to Item 1.c of Definition 6.4.1, $v \in \mathsf{q}^{-1}(p)$. However, $\mathsf{r} = \mathsf{q} \cap \mathsf{g}$, so, $v \in \mathsf{q}^{-1}(p) \cap \mathsf{q}^{-1}(p) = \mathsf{r}^{-1}(p)$, which is impossible. As a consequence of this inclusion together with Point *(12)*, we have that *(13)* $p^\star = \mathsf{max}(\mathsf{rng}(\mathsf{q}^\star \restriction \mathrm{A})) < p$. Now, due to Points *(3)* and *(13)*, it holds that *(14)* $\mathsf{g}^{\star(>p)} = \mathsf{g}^{(>p)}$, *(15)* $\mathsf{g}^{\star -1}(p) = \mathsf{g}^{-1}(p) \setminus \mathrm{A}$, *(16)* $\mathsf{dom}\Big(\mathsf{g}^{\star(<p)}\Big) = \mathsf{g}^{\star -1}(p^\star) = \mathrm{A}$, and *(17)* $\mathsf{dom}\Big(\mathsf{g}^{\star(<p^\star)}\Big) = \emptyset$. Observe that, by Point *(14)*, g and $\mathsf{g}^\star$ only differ on the priorities p and $p^\star$. In particular, $\mathsf{rng}(\mathsf{g}^\star) = \mathsf{rng}(\mathsf{g}) \cup \{p^\star\}$. Thus, by Item *(i)* of the definition of subgame function and Points *(8)* and *(13)*, we have that $\mathsf{rng}(\mathsf{g}^\star) \subseteq \mathsf{rng}(\mathsf{pr}_\partial)$. By Points *(15)* and *(16)*, *(18)* $\partial^{\leq p}_{\mathsf{g}^\star} = \partial^{\leq p}_{\mathsf{g}}$ is

obviously a game. Moreover, due to Point *(4)*, it holds that *(19)* $\mathsf{dom}\big(\mathsf{g}^{\star(\leq p^\star)}\big) = \mathrm{A} \subset \mathsf{dom}\big(\mathsf{g}^{\star(\leq p)}\big)$. Finally, $\partial_{\mathsf{g}^\star}^{\leq p^\star}$ is a game, by Point *(12)*. Summing up, it follows that *(20)* $\mathsf{g}^\star$ is a subgame function.

We can now prove the tree parts in which Item 1 of Definition 6.4.1 applied to $(\mathsf{q}^\star, \mathsf{g}^\star)$ is split. First notice that, since $\mathsf{r} = \mathsf{q} \cap \mathsf{g}$, we have that *(21)* $\mathsf{r}^{\star(>p)} = \mathsf{r}^{(>p)}$, due to Points *(5)* and *(14)*. Moreover, *(22)* $\mathsf{r}^{\star -1}(p^\star) \subseteq \mathsf{pr}_\partial^{-1}(p^\star)$, by Point *(8)*, and *(23)* $\mathsf{r}^{\star(<p^\star)} = \varnothing$, by Point *(17)*. Finally, by Points *(10)*, *(13)*, and *(15)* and the observation that $\mathrm{P} \cap \mathsf{g}^{-1}(p) = \emptyset$, it holds that *(24)* $\mathsf{r}^{\star -1}(p) = \mathrm{R}$. At this point, Item 1.a of the definition of state space follows from Points *(21)* and *(24)*, since R is $\wp$-maximal in ∂_s due to Definition 6.4.2. By putting together Points *(18)-(23)* with Point *(12)*, also Item 1.b can be derived. To prove that Item 1.c, *i.e.*, $\mathsf{q}^\star = \mathsf{pr}_\partial \uplus \mathsf{r}^\star$, one can use Points *(5)* and *(21)* for the priorities greater than p, Points *(10)* and *(24)* for the priority p, and Point *(8)* for the priorities smaller than p.

In the end, by Points *(11)* and *(20)*, and the fact that Item 1 of Definition 6.4.1 holds on $(\mathsf{q}^\star, \mathsf{g}^\star)$, it follows that $(\mathsf{q}^\star, \mathsf{g}^\star) \in \mathrm{S}_\partial$.

To conclude this case of the proof, we need just to show that $(\mathsf{q}^\star, \mathsf{g}^\star) \prec_\partial (\mathsf{q}, \mathsf{g})$. If $\mathrm{R} \subseteq \mathsf{q}^{-1}(p)$, by Point *(1)* and *(13)*, we have that $\mathsf{q}^\star = \mathsf{q}$ and $p^\star < p$. Thus, the thesis is derived from Item 3.b of Definition 6.4.1. If, on the other hand, $\mathrm{R} \setminus \mathsf{q}^{-1}(p) \neq \emptyset$, due to Point *(5)* and *(7)*, the thesis follows from Item 3.a of the same definition.

Consider now the case in which the set R is $\wp$-closed in ∂_s. By Lines 1, 5-7 of Algorithm 10, we have that *(25)* $p^\star = \mathsf{bep}_\partial^{\overline{\wp}}(\mathrm{R}, \mathsf{q})$, *(26)* $\mathsf{q}^\star = \mathsf{pr}_\partial \uplus \mathsf{q}^{(\geq p^\star)}[\mathrm{R} \mapsto p^\star]$, and *(27)* $\mathsf{g}^\star = \mathsf{g}[\mathsf{dom}\big(\mathsf{g}^{(<p^\star)}\big) \mapsto p^\star]$. Due to Point *(25)*, the definition of best escape priority, and the fact that R is $\wp$-closed in ∂_s, it is not hard to see that *(28)* $p^\star > p$. Moreover, due to Item *(ii)* of the definition of g stratified region function, it holds that *(29)* $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$, since the moves escaping from R can only reach positions in $\mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$.

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

Moreover, by Point *(26)*, it holds that *(34)* $\mathsf{q}^{\star(>p^\star)} = \mathsf{q}^{(>p^\star)}$, *(35)* $\mathsf{q}^{\star -1}(p^\star) = \mathsf{q}^{-1}(p^\star) \cup \mathrm{R}$, and *(36)* $\mathsf{q}^{\star(<p^\star)} \subseteq \mathsf{pr}_\partial{}^{(<p^\star)}$. Consequently, *(37)* $\mathsf{r}^{\star(>p^\star)} = \mathsf{r}^{(>p^\star)}$, by Points *(30)* and *(34)*, and *(38)* $\mathsf{r}^{\star(<p^\star)} = \emptyset$, by Point *(31)*. In addition, due to Points *(32)* and *(36)*, together with Item *(ii.b)* of the definition of subgame function and Proposition 6.3.1, we have that *(39)* $\mathsf{r}^{\star -1}(p^\star) = \mathsf{q}^{\star -1}(p^\star) \cap \mathsf{g}^{\star -1}(p^\star) = (\mathsf{q}^{-1}(p^\star) \cup \mathrm{R}) \cap \mathsf{g}^{\star -1}(p^\star) = \mathsf{r}^{-1}(p^\star) \cup \mathrm{R}$ is an $\wp$-region in $\partial_{s^\star}$. At this point, Item 1 of Definition 6.4.1 follows by simply verifying that all requirements are satisfied.

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

In the end, by Points *(33)* and *(40)*, and the fact that Item 1 of Definition 6.4.1 holds on $(\mathsf{q}^\star, \mathsf{g}^\star)$, it follows that $(\mathsf{q}^\star, \mathsf{g}^\star) \in \mathrm{S}_\mathfrak{D}$.

To conclude, as for the previous case, it remains to show that $(\mathsf{q}^\star, \mathsf{g}^\star) \prec_\mathfrak{D} (\mathsf{q}, \mathsf{g})$. This fact easily follows from Item 3.a of Definition 6.4.1, where the priority p is set to $p^\star$. Indeed, by Definition 6.4.2, we have that $\emptyset \neq \mathrm{R} \subseteq \mathrm{Ps}_{\mathfrak{D}_s}$, so, $\mathsf{q}^{-1}(p^\star) \cap \mathrm{R} = \emptyset$, since $p^\star > p$ as stated in Point *(28)*. Therefore, by Point *(31)*, it follows that $\mathsf{q}^{-1}(p^\star) \subset \mathsf{q}^{-1}(p^\star) \cup \mathrm{R} = \mathsf{q}^{\star\,-1}(p^\star)$. □

6.5 Experimental Evaluation

In this section we shall try to assess the effectiveness of the proposed approach. To this end, the priority promotion technique has been implemented in the tool PGSolver [FL09]. The tool, written in OCaml, collects implementations of several parity game solvers proposed in the literature and provides benchmarking tools, which can generate different forms of parity games. The available benchmarks divide into concrete and synthetic problems. The *concrete benchmarks* encode validity and verification problems for temporal logics. They consist in parity games resulting from encodings of the language inclusion problem between automata, specifically a non-deterministic Büchi automaton and a deterministic one, simple reachability problems, the Tower of Hanoi problems, and a fairness verification problem, the Elevator problem (see [FL09] for more details on this benchmarks). The *synthetic benchmarks* divide into randomly generated games and various families, corresponding to difficult cases, such as clique and ladder-like games, and worst cases for the solvers implemented in PGSolver. To fairly compare the different solution techniques used by the underlying algorithms, the solvers involved in the experiments have been isolated from the generic solver implemented in PGSolver, which exploits few game transformation and decomposition techniques in the attempt to speed up the solution process. Indeed, those optimizations can, in some cases, solve the game without even calling the selected algorithm, and, in other cases, the resulting overhead can even outweigh the solver time, making the comparison among solvers virtually worthless [FL09]. Experiments were also conducted with different optimizations enabled and the results exhibit patterns similar to the ones emerging in the following experimental evaluation, even though solution times and the gap among solvers may reduce considerably in some cases, depending on the benchmarks considered.

The algorithms considered in the experimentation are the Zielonka algorithm *Rec* [Zie98], its two dominion decomposition variants, *Dom* [JPZ06, JPZ08] and *Big* [Sch07], the strategy improvement algorithm *Str* [VJ00][1], the small progress measure algorithm *SmPr* [Jur00], and the one of this article,

[1]Note that the version of small-progress measure used in the experiments, which is included in the official release of

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

Table 6.2: Execution times (in seconds) on several benchmark families. Time out (†) is set to 600 seconds and memory out (‡) to 7.5Gb.

PP (available at `https://github.com/tcsprojects/pgsolver`).[2]

6.5.1 Special Families

Table 6.2 displays the results of all the solvers involved on the benchmark families available in PGSolver. We only report on the biggest instances we could deal with, given the available computational resources[3]. The parameter Positions refers to the number of positions in the games and the best results are emphasized in bold. The first three sections of the table, each comprising three instances of increasing size for the same benchmark, consider the concrete verification problems mentioned above. On all the instances of the Tower of Hanoi problem most of the solvers perform reasonably well, except for *SmPr* and for

PGSolver, is not the original one proposed by Jurdzinski, as it performs a progress interleaving for both players at once, instead of focusing on a single player like the original algorithm. According to the authors of PGSolver, this optimization allows for a non-trivial performance improvement.

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

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

Str. The Elevator problem, instead, proved to be very demanding, both in terms of time and memory, for all the solvers, except for our new algorithm and for *Dom*, which, however, could not solve the biggest instance within the time limit of 10 minutes. Our solver performs extremely well on both this benchmarks and on the instances of the Language Inclusion problem, whose biggest instance could be solved only by *Rec* among the other solvers. On the worst case benchmarks, PP performs quite well also on Ladder, Strategy Improvement, Clique, and Jurdziński games, all of which proved to be considerably difficult for all the other solvers. The Modelchecker game is a tie with *Rec*. On the Recursive Ladder game PP is the only algorithm that could solve the instance used in the experiments. However, the even instances of that game, unlike the odd ones, turn out to be very easy to solve by *Str*, which requires less than a second even for the instance with $5 \cdot 10^5$ positions.
The reason is that those instances are completely won by player odd and, in addition, since *Str* looks for a winning strategy of player even, it immediately discovers that the initial even strategy cannot be improved further. On the other hand, the odd instances are completely won by player even and the algorithm requires a linear number of iterations to find the winning strategy. The new solver exhibits the most consistent behavior overall on these benchmarks, significantly outperforming the other solvers considered in the experiments. Moreover, for all of them the priority promotion algorithm requires no promotions regardless of the input parameters, except for the Elevator problem, on which it performs only two. As a consequence, PP requires polynomial time on all the benchmarks reported in the table.

Table 6.3, instead, reports a comparison with the quasi-polynomial time algorithms proposed in [FJS+17] and [JL17]. We considered a preliminary C++ implementation of the former, *QPT*, made available by the authors themselves (the iOS executable can be obtained at `https://cgi.csc.liv.ac.uk/~dominik/parity`), and an OCaml implementation of the latter, *SPM*, recently included in PGSolver[4]. Despite the better worst case upper bound and the theoretical relevance of the result, neither algorithm could solve the benchmark instances reported in Table 6.2. Hence, we had to resort to much smaller instances. The comparison may not be particularly meaningful, given the differences in implementation, neither fair towards the solvers implemented in OCaml. It suggests, nonetheless, that the practical effectiveness of the two quasi-polynomial solvers, at least of the versions available at the time of writing, is very far from that of the exponential algorithms and, in particular, of PP. The table shows indeed a considerable gap, often of many orders of magnitude, which may only to some extent be explained by a possible lack of optimization.

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

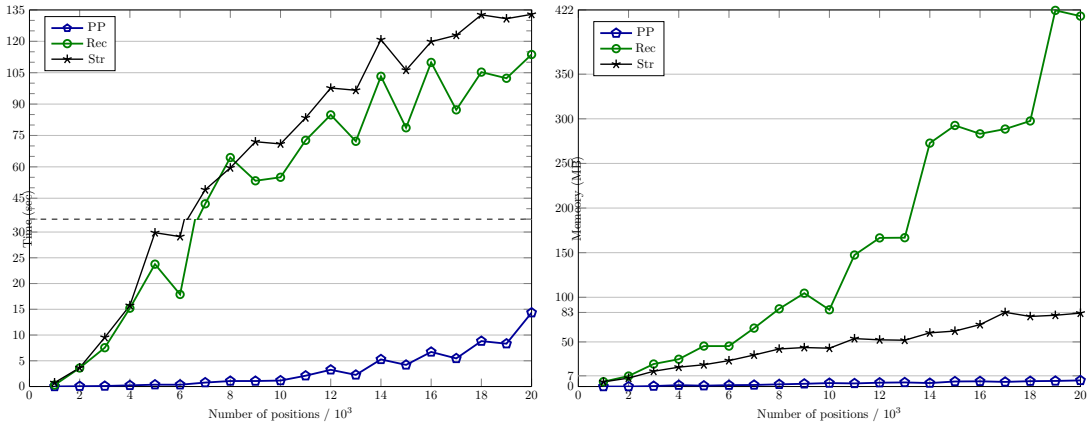


Figure 6.4: Time and auxiliary memory on random games with 2 moves per position.

6.5.2 Random Games

Figure 6.4 compares the running times (left-hand side) and memory requirements (right-end side) of the new algorithm PP against *Rec* and *Str* on 2000 random games of size ranging from 5000 to 20000 positions and 2 outgoing moves per position. Interestingly, these random games proved to be quite challenging for all the considered solvers. We set a time-out to 180 seconds (3 minutes). Both *Dom* and *Big* perform quite poorly on those games, hitting the time-out already for very small instances, and we decided to leave them out of the picture. The behavior of the solvers is typically highly variable even on games of the same size and priorities. To summaries the results, the average running time on clusters of games seemed the most appropriate choice in this case. Therefore, each point in the graph shows the average time over a cluster of 100 different games of the same size: for each size value n, we chose a number $k = n \cdot i/10$ of priorities, with $i \in [1, 10]$, and 10 random games were generated for each pair of n and k. The new algorithm perform significantly better than the others on those games. The right-hand side graph also shows that the theoretical improvement on the auxiliary memory requirements of the new algorithm has a considerable practical impact on memory consumption compared to the other solvers. In these particular benchmarks, the additional memory required by PP amounts to 7 Mb on average for the biggest instances, while *Str* requires up to 83 Mb and *Rec* up to 422 Mb.

Table 6.4 reports on some experiments on random games with higher number of moves per position. The resulting games turn out to be much easier to solve for most the solvers, in particular for *Rec*, *Dom*, and PP. The benchmarks here are divided into 5 cluster with increasing number of positions from 10^4 to 10^5. Each cluster contains 120 games each, with number of priorities linear on the number of positions and 10 to 100 moves per position. On each row, the table reports the average time required for the

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

Table 6.4: Average execution time (in seconds) on random games with 10 and 100 moves per position. Time out is set to 60 seconds.

solution over the 120 games of each cluster. The last row, instead, details the percentage of games on which the corresponding solver could not terminate before the timeout set to 60 seconds. The higher number of moves significantly increases the dimension of the regions computed by PP and, consequently, also the chances for it to find a closed one. Indeed, the number of promotions required by the algorithm on all those games is typically below half a dozen. The whole solution time is almost exclusively due to a very limited number of attractors needed to compute the few regions contained in the games. The only other solvers that can easily solve all these games within 60 seconds are *Rec*, whose performance is only slightly worse than that of PP, and *Dom*, which could solve almost all of the games before the timeout.

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

Table 6.5: Average execution time (in seconds) on random games with logarithmic number of priorities. Time out is set to 60 seconds.

Table 6.5 shows an experimental comparison on random games where the number of priorities grows logarithmically *w.r.t.* the number of positions. The relevance of these benchmarks stems from the fact that in practical applications, such as verification problems, the resulting encodings into parity games

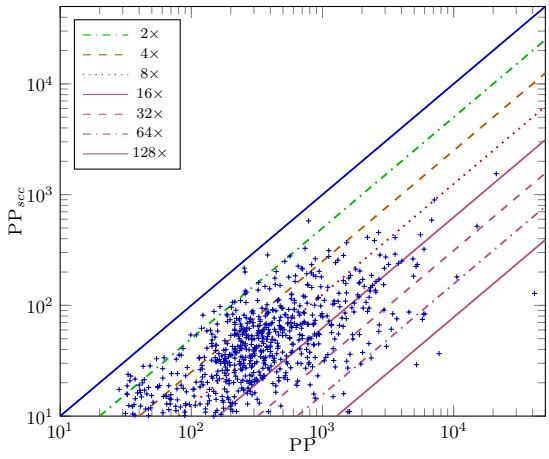


Figure 6.5: Comparison between PP and PP_{scc} on random games with 50000 positions.

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

6.5.3 PP with subgame decomposition

Finally, Figure 6.5 compares the performance of the subgame decomposition version of PP with the original algorithm. Here, the decomposition schema has been realized by instantiating function F in Algorithm 10 with an SCC decomposition procedure that computes some minimal SCC in the current subgame. We refer to resulting procedure with PP_{scc}. In order to assess its effectiveness, we used a new pool of benchmarks that contains 740 games, each with 50000 positions, 2 moves per positions and priorities varying from 8000 to 12000. These games are much harder than the ones reported in Figure 6.4 and have been specifically selected among random games whose solution requires PP more 30 seconds and up to more than 10 hours to be solved. The results, drawn on a logarithmic scale, show that the decomposition technique, described in Section 6.4, does pay off significantly. It can run up to 128 times faster than the original version, reducing the solution time from several hours to a few seconds in some

cases.

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

Table 6.3: Execution times (in seconds) on several benchmark families. Time out (†) is set to 600 seconds.

Very similar results can be obtained by instantiating function F in Algorithm 10 with a simpler reachability computation procedure that collects all the positions reachable from some arbitrary position in the current game. The resulting technique, called PP_{rch}, cannot, in general, provide as deep a decomposition as does the SCC-based decomposition. The additional overhead of the latter one is, however, usually higher. As shown in Figure 6.6, the two algorithm are not comparable, in the sense that there is no clear winner between the two: on games with more structure the SCC decomposition usually performs better, being able to compute smaller subgames; on games with less structure, the reachability decomposition technique obtains sufficiently small subgames with less overhead.

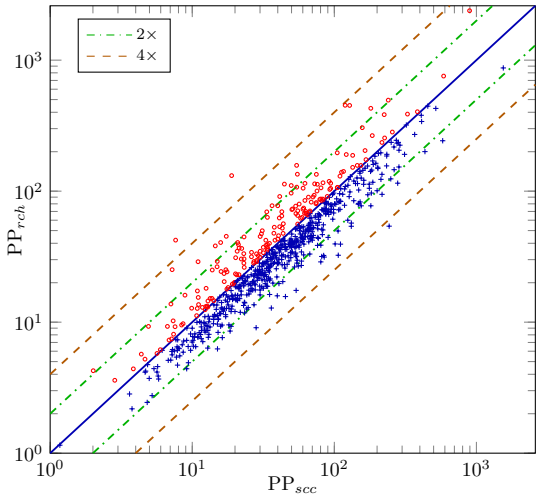


Figure 6.6: Comparison between PP_{scc} and PP_{rch} on random games with 50000 positions.

Chapter 7

A Delayed Promotion Policy for Parity Games

This chapter contains the first refinement of the Priority Promotion approach, obtained by applying a different promotion policy. Even if the structure of the solver is the same as the original algorithm, it is shown that the number of iteration required to solve a game is strongly dependent on the policy that determines the order of promotions. We also present an intermediate version of the solver which can be considered as an optimized PP algorithm.

This work has been presented at the

- 7th International Symposium on Games, Automata, Logics and Formal Verification (GANDALF'16), held in September 14-16, 2016, in Catania, Italy, with the title of "A Delayed Promotion Policy for Parity Games" and published on Volume 226 of EPTCS, pages 30-45, 2016.

The content of the chapter is based on the journal version of [BDM16a] published in Information and Computation [BDM18a].

7.1 Abstract

We continue the study of the priority promotion approaches trying to find a remedy to its exponential behavior. We propose a new algorithm, called DP, built on top of a slight variation of PP, called PP+. The PP+ algorithm simply avoids resetting previous promotions to quasi dominions of the same parity.

In this case, indeed, the relevant properties of those quasi dominions are still preserved. This variation enables the new DP promotion policy, that delays promotions that require a reset and only performs those leading to the highest quasi dominions among the available ones. Experiments on randomly generated games show that the new approach performs much better than PP in practice, while still preserving the same space complexity.

7.2 The Priority Promotion Approach

The priority promotion approach proposed in [BDM18b] attacks the problem of solving a parity game $\mathcal{D}$ by computing one of its dominions D, for some player $\wp \in \{0,1\}$, at a time. Indeed, once the $\wp$-attractor $\mathrm{D}^\star$ of D is removed from $\mathcal{D}$, the smaller game $\mathcal{D} \setminus \mathrm{D}^\star$ is obtained, whose positions are winning for one player iff they are winning for the same player in the original game. This allows for decomposing the problem of solving a parity game into iteratively finding its dominions [JPZ08].

In order to solve the dominion problem, the idea is to start from a much weaker notion than that of dominion, called *quasi dominion*. Intuitively, a quasi $\wp$-dominion is a set of positions on which player $\wp$ has a strategy whose induced plays either remain inside the set forever and are winning for $\wp$ or can exit from it passing through a specific set of escape positions.

Definition 7.2.1 (Quasi Dominion)**.** *Let $\mathcal{D} \in \mathcal{P}$ be a game and $\wp \in \{0,1\}$ a player. A non-empty set of positions* $\mathrm{Q} \subseteq \mathrm{Ps}$ *is a* quasi $\wp$-dominion *in $\mathcal{D}$ if there exists an $\wp$-strategy $\sigma_\wp \in \mathrm{Str}^\wp(\mathrm{Q})$ such that, for all $\overline{\wp}$-strategies $\sigma_{\overline{\wp}} \in \mathrm{Str}^{\overline{\wp}}(\mathrm{Q})$, with $\mathsf{int}^{\overline{\wp}}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\wp}})$, and positions $v \in \mathrm{Q}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ satisfies $\mathsf{pr}(\pi) \equiv_2 \wp$, if π is infinite, and $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\wp}}(\mathrm{Q})$, otherwise.*

Observe that, if all the induced plays remain in the set Q forever, this is actually an $\wp$-dominion and, therefore, a subset of the winning region $\mathrm{Wn}_\wp$ of $\wp$. In this case, the escape set of Q is empty, *i.e.*, $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q}) = \emptyset$, and Q is said to be *$\wp$-closed*. In general, however, a quasi $\wp$-dominion Q that is not an $\wp$-dominion, *i.e.*, such that $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q}) \neq \emptyset$, need not to be a subset of $\mathrm{Wn}_\wp$ and it is called *$\wp$-open*. Indeed, in this case, some induced play may not satisfy the winning condition for that player once exited from Q, by visiting a cycle containing a position with maximal priority of parity $\overline{\wp}$. The set of pairs $(\mathrm{Q}, \wp) \in 2^{\mathrm{Ps}} \times \{0,1\}$, where Q is a quasi $\wp$-dominion, is denoted by QD, and is partitioned into the sets QD^- and QD^+ of open and closed quasi $\wp$-dominion pairs, respectively.

Algorithm 11: Parity Game Solver.

signature $\mathsf{sol}_\Gamma : \mathcal{P} \to_\ni (2^{\mathrm{Ps}_\ni} \times 2^{\mathrm{Ps}_\ni})$
function $\mathsf{sol}_\Gamma[\ni]$
1 **if** $\mathrm{Ps}_\ni = \emptyset$ **then**
2 **return** $(\emptyset, \emptyset)$
else
3 $(\mathrm{R}, \wp) \leftarrow \mathsf{src}_\Gamma(\ni)$
4 $\mathrm{R}^\star \leftarrow \mathsf{atr}^\wp_\ni[\mathrm{R}]$
5 $(\mathrm{W}_0', \mathrm{W}_1') \leftarrow \mathsf{sol}_\Gamma[\ni \setminus \mathrm{R}^\star]$
6 $(\mathrm{W}_\wp, \mathrm{W}_{\overline{\wp}}) \leftarrow (\mathrm{W}'_\wp \cup \mathrm{R}^\star, \mathrm{W}'_{\overline{\wp}})$
7 **return** $(\mathrm{W}_0, \mathrm{W}_1)$

Algorithm 12: The Searcher.

signature $\mathsf{src}_\Gamma : \mathcal{P} \to_\ni \mathrm{Rg}^+_\ni$
function $\mathsf{src}_\Gamma(\ni)$
1 **return** $\mathsf{src}_{\Gamma(\ni)}(\top_{\Gamma(\ni)})$

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

The priority promotion algorithm explores a partial order $\mathcal{D}$, whose elements, called *states*, record information about the open quasi dominions computed along the way. Different partial orders can be associated to the same game. Therefore, we shall implicitly assume a function Γ mapping every game $\ni$ to a partial order $\mathcal{D} = \Gamma(\ni)$, where $\ni_\mathcal{D} \triangleq \ni$. The initial state of the search is the top element of the order, where the quasi dominions are initialized to the sets of positions with the same priority. At each step, a new quasi dominion is extracted from the current state, by means of a *query* operator $\Re$, and used to compute a successor state, by means of a *successor* operator $\downarrow$, if the quasi dominion is open. If, on the other hand, it is closed, the search is over. Algorithm 12 implements the dominion search procedure $\mathsf{src}_\mathcal{D}$. A *compatibility relation* $\succ$ connects the query and the successor operators. The relation holds between states of the partial order and the quasi dominions that can be extracted by the query operator. Such a relation defines the domain of the successor operator. The partial order together with the query and successor operators and the compatibility relation form what is called a *dominion space*.

Definition 7.2.2 (Dominion Space)**.** *A* dominion space *for a game $\ni \in \mathcal{P}$ is a tuple $\mathcal{D} \triangleq \langle \ni, \mathcal{S}, \succ, \Re, \downarrow \rangle$, where* (1) *$\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$ is a well-founded partial order* w.r.t. *$\prec \subset \mathrm{S} \times \mathrm{S}$ with distinguished element $\top \in \mathrm{S}$,* (2) *$\succ \subseteq \mathrm{S} \times \mathrm{QD}^-$ is the* compatibility relation*,* (3) *$\Re : \mathrm{S} \to \mathrm{QD}$ is the* query operator *mapping each element $s \in \mathrm{S}$ to a quasi dominion pair $(\mathrm{Q}, \wp) \triangleq \Re(s) \in \mathrm{QD}$ such that, if $(\mathrm{Q}, \wp) \in \mathrm{QD}^-$ then $s \succ (\mathrm{Q}, \wp)$,* and (4) *$\downarrow : \succ \to \mathrm{S}$ is the* successor operator *mapping each compatible pair $(s, (\mathrm{Q}, \wp)) \in \succ$ to the element $s^\star \triangleq s{\downarrow}(\mathrm{Q}, \wp) \in \mathrm{S}$ with $s^\star \prec s$.*

The notion of dominion space is quite general and can be instantiated in different ways, by providing specific query and successor operators. In [BDM18b], indeed, it is shown that the search procedure $\mathsf{src}_{\mathcal{D}}$ is sound and complete on any dominion space $\mathcal{D}$. In addition, its time complexity is linear in the *execution depth* of the dominion space, namely the length of the longest chain in the underlying partial order compatible with the successor operator, while its space complexity is only logarithmic in the space *size*, since only one state at the time needs to be maintained. A specific instantiation of dominion space, called PP *dominion space*, is the one proposed and studied in [BDM18b]. Here we propose a different one, starting from a slight optimization, called PP+, of that original version.

7.2.1 PP+ Dominion Space

In order to instantiate a dominion space, we need to define a suitable query function to compute quasi dominions and a successor operator to ensure progress in the search for a closed dominion. The priority promotion algorithm proceeds as follows. The input game is processed in descending order of priority. At each step, a subgame of the entire game, obtained by removing the quasi dominions previously computed at higher priorities, is considered. At each priority of parity $\wp$, a quasi $\wp$-dominion Q is extracted by the query operator from the current subgame. If Q is closed in the entire game, the search stops and returns Q as result. Otherwise, a successor state in the underlying partial order is computed by the successor operator, depending on whether Q is open in the current subgame or not. In the first case, the quasi $\wp$-dominion is removed from the current subgame and the search restarts on the new subgame that can only contain positions with lower priorities. In the second case, Q is merged together with some previously computed quasi $\wp$-dominion with higher priority. Being a dominion space well-ordered, the search is guaranteed to eventually terminate and return a closed quasi dominion. The procedure requires the solution of two crucial problems: *(a) extracting a quasi dominion* from a subgame and *(b) merging together two quasi $\wp$-dominions* to obtain a bigger, possibly closed, quasi $\wp$-dominion.

The solution of the first problem relies on the definition of a specific class of quasi dominions, called *regions*. An $\wp$-region R of a game ∂ is a special form of a quasi $\wp$-dominion of ∂ with the additional requirement that all the escape positions in $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$ have the maximal priority $p \triangleq \mathsf{pr}(\partial) \equiv_2 \wp$ in ∂. In this case, we say that the $\wp$-region R has priority p. As a consequence, if the opponent $\overline{\wp}$ can escape from the $\wp$-region R, he must visit a position with the highest priority in it, which is of parity $\wp$.

Definition 7.2.3 (Region)**.** *A quasi $\wp$-dominion* R *is an $\wp$-region in ∂ if* $\mathsf{pr}(\partial) \equiv_2 \wp$ *and all the positions in* $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$ *have priority* $\mathsf{pr}(\partial)$*,* i.e. $\mathsf{esc}^{\overline{\wp}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}(\mathsf{pr}(\partial))$*.*

It is important to observe that, in any parity game, an $\wp$-region always exists, for some $\wp \in \{0, 1\}$. In

particular, the set of positions of maximal priority in the game always forms an $\wp$-region, with $\wp$ equal to the parity of that maximal priority. In addition, the $\wp$-attractor of an $\wp$-region is always an $\wp$-maximal $\wp$-region. A closed $\wp$-region in a game is clearly an $\wp$-dominion in that game. These observations give us an easy and efficient way to extract a quasi dominion from every subgame: collect the $\wp$-attractor of the positions with maximal priority p in the subgame, where $p \equiv_2 \wp$, and assign p as priority of the resulting region R. This priority, called *measure* of R, intuitively corresponds to an under-approximation of the best priority player $\wp$ can force the opponent $\overline{\wp}$ to visit along any play exiting from R.

Proposition 7.2.1 (Region Extension [BDM18b])**.** *Let $\partial \in \mathcal{P}$ be a game and $\mathrm{R} \subseteq \mathrm{Ps}$ an $\wp$-region in ∂. Then, $\mathrm{R}^\star \triangleq \mathsf{atr}^\wp[\mathrm{R}]$ is an $\wp$-maximal $\wp$-region in ∂.*

A solution to the second problem, the merging operation, is obtained as follows. Given an $\wp$-region R in some game ∂ and an $\wp$-dominion D in a subgame of ∂ that does not contain R itself, the two sets are merged together, if the only moves exiting from $\overline{\wp}$-positions of D in the entire game lead to higher priority $\wp$-regions and R has the lowest priority among them. The priority of R is called the *best escape priority* of D for $\overline{\wp}$. The correctness of this merging operation is established by the following proposition.

Proposition 7.2.2 (Region Merging [BDM18b])**.** *Let $\partial \in \mathcal{P}$ be a game, $\mathrm{R} \subseteq \mathrm{Ps}$ an $\wp$-region, and $\mathrm{D} \subseteq \mathrm{Ps}_{\partial \setminus \mathrm{R}}$ an $\wp$-dominion in the subgame $\partial \setminus \mathrm{R}$. Then, $\mathrm{R}^\star \triangleq \mathrm{R} \cup \mathrm{D}$ is an $\wp$-region in ∂. Moreover, if both R and D are $\wp$-maximal in ∂ and $\partial \setminus \mathrm{R}$, respectively, then $\mathrm{R}^\star$ is $\wp$-maximal in ∂ as well.*

The merging operation is implemented by promoting all the positions of $\wp$-dominion D to the measure of R, thus improving the measure of D. For this reason, it is called a *priority promotion*. In [BDM18b] it is shown that, after a promotion to some measure p, the regions with measure lower than p might need to be destroyed, by resetting all the contained positions to their original priority. This necessity derives from the fact that the new promoted region may attract positions from lower ones, thereby potentially invalidating their status as regions. Indeed, in some cases, the player that wins by remaining in the region may even change from $\wp$ to $\overline{\wp}$. As a consequence, the reset operation is, in general, unavoidable. The original priority promotion algorithm applies the reset operation to all the lower priority regions. However, the following property ensures that this can be limited to the regions belonging to the opponent player only.

Proposition 7.2.3 (Region Splitting)**.** *Let $\partial^\star \in \mathcal{P}$ be a game and $\mathrm{R}^\star \subseteq \mathrm{Ps}_{\partial^\star}$ an $\wp$-maximal $\wp$-region in $\partial^\star$. For any subgame ∂ of $\partial^\star$ and $\wp$-region $\mathrm{R} \subseteq \mathrm{Ps}$ in ∂, if $\mathrm{R}^\natural \triangleq \mathrm{R} \setminus \mathrm{R}^\star \neq \emptyset$, then $\mathrm{R}^\natural$ is an $\wp$-region in $\partial \setminus \mathrm{R}^\star$.*

Proof. Let us first prove that $\mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right) \subseteq \mathsf{esc}^{\overline{\wp}}(\mathrm{R})$. Assume, by contradiction, that there exists a position $v \in \mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$ which does not belong to $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$. Let us consider the case where v is an $\wp$-position of $\mathrm{R}^\sharp$ first. Then, one of its moves leads to R in ∂. If such move leads to $\mathrm{R}^\sharp$, then it does so also in the subgame $\partial \setminus \mathrm{R}^\star$, and v cannot belong to the escape set of $\mathrm{R}^\sharp$. If, on the other hand, it leads to $\mathrm{R}^\star$, then, due to the maximality of $\mathrm{R}^\star$, v must belong to $\mathrm{R}^\star$ as well, contradicting the assumption that $v \in \mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$. Assume now that v is a $\overline{\wp}$-position. Then, all of its moves lead to R in ∂. Clearly, all the moves from v that remain in the subgame $\partial \setminus \mathrm{R}^\star$, if any, must lead to $\mathrm{R}^\sharp$. Hence, v cannot be an escaping position of $\mathrm{R}^\sharp$ in $\partial \setminus \mathrm{R}^\star$ in this case either. As a consequence, all the positions in $\mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$ have maximal priority in ∂ and, *a fortiori*, in $\partial \setminus \mathrm{R}^\star$ as well.

Let us now prove that $\mathrm{R}^\sharp$ is a quasi $\wp$-dominion. Let $\sigma \in \mathrm{Str}^{\wp}(\mathrm{R})$ be the witness $\wp$-strategy for R as in Definition 7.2.1. Hence, regardless of the $\overline{\wp}$-strategy used by the opponent, each play induced by σ, either is infinite, hence remains in R forever, and is winning for $\wp$, or is finite and ends in some position of the escape set $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$. Consider now the restriction $\sigma' \triangleq \sigma{\restriction}\mathrm{R}^\sharp$ of σ to the $\wp$-positions of $\mathrm{R}^\sharp$. Let π be a play induced by σ'. If π remains in $\mathrm{R}^\sharp \subseteq \mathrm{R}$ forever, then it is clearly winning for $\wp$, as that play is also compatible with σ. If, on the other hand, it is finite, then it must end in some position $v \in \mathrm{R}^\sharp$. If v is a $\overline{\wp}$-position, then it has a move leaving $\mathrm{R}^\sharp$ and, therefore, must belong to the escape $\mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$. Assume v is an $\wp$-position. If v is not in the domain of σ', then it is not in the domain of σ either. Hence, it must belong to $\mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{R})$. Being $\mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right) \subseteq \mathsf{esc}^{\overline{\wp}}_{\partial}(\mathrm{R})$ and $v \in \mathrm{R}^\sharp$, we can conclude that $v \in \mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$. Finally, assume, by contradiction, that v is in the domain of σ'. Then, $\sigma'(v) = \sigma(v) \in \mathrm{R}$. Since π ends in v, however, $\sigma'(v) \notin \mathrm{R}^\sharp$. But then, v would be an $\wp$-position in R with a move leading to $\mathrm{R}^\star$, contradicting the $\wp$-maximality of $\mathrm{R}^\star$. We can, thus, conclude that v must belong to $\mathsf{esc}^{\overline{\wp}}_{\partial\setminus \mathrm{R}^\star}\left(\mathrm{R}^\sharp\right)$. □

This proposition, together with the observation that $\overline{\wp}$-regions that can be extracted from the corresponding subgames cannot attract positions contained in any retained $\wp$-region, allows for preserving all the lower $\wp$-regions computed so far.

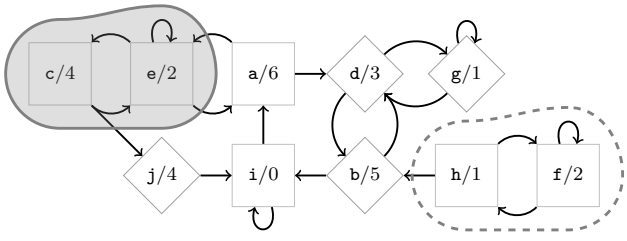


Figure 7.1: Running example.

	1	2	3	4	5	6
6	$\mathtt{a}\downarrow$	···	···	···	···	$\mathtt{a},\mathtt{b},\mathtt{d},\mathtt{g},\mathtt{i},\mathtt{j}\downarrow$
5	$\mathtt{b},\mathtt{f},\mathtt{h}\downarrow$	···	···	$\mathtt{b},\mathtt{d},\mathtt{f},\mathtt{g},\mathtt{h}\downarrow$	···	
4	$\mathtt{c},\mathtt{j}\downarrow$	$\mathtt{c},\mathtt{e},\mathtt{j}\downarrow$	···	$\mathtt{c},\mathtt{j}\downarrow$	$\mathtt{c},\mathtt{e},\mathtt{j}\downarrow$	$\mathtt{c},\mathtt{e}\uparrow_6$
3	$\mathtt{d}\downarrow$	$\mathtt{d}\downarrow$	$\mathtt{d},\mathtt{g}\uparrow_5$			
2	$\mathtt{e}\uparrow_4$			$\mathtt{e}\uparrow_4$		
1		$\mathtt{g}\uparrow_3$				
0					$\mathtt{i}\uparrow_6$	

Table 7.1: PP+ simulation.

Example 7.2.1. *To exemplify the idea, Table 7.1 shows a simulation of the resulting procedure on the parity game of Figure 7.1, where diamond shaped positions belong to player* 0 *and square shaped ones to its opponent* 1*. Player* 0 *wins the entire game, hence the* 0*-region containing all the positions is a* 0*-dominion in this case. Each cell of the table contains a computed region. A downward arrow denotes a region that is open in the subgame where it is computed, while an upward arrow means that the region gets to be promoted to the priority in the subscript. The index of each row corresponds to the measure of the region. Following the idea sketched above, the first region obtained is the single-position* 0*-region* $\{\mathtt{a}\}$ *of measure 6, which is open because of the two moves leading to* $\mathtt{d}$ *and* $\mathtt{e}$*. The open* 1*-region* $\{\mathtt{b},\mathtt{f},\mathtt{h}\}$ *of measure 5 is, then, formed by attracting both* $\mathtt{f}$ *and* $\mathtt{h}$ *to* $\mathtt{b}$*, which is open in the subgame where* $\{\mathtt{a}\}$ *is removed. Similarly, the* 0*-region* $\{\mathtt{c},\mathtt{j}\}$ *of measure* 4 *and the* 1*-region* $\{\mathtt{d}\}$ *of measure* 3 *are open, once removed* $\{\mathtt{a},\mathtt{b},\mathtt{f},\mathtt{h}\}$ *and* $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{f},\mathtt{h}\}$*, respectively, from the game. At priority* 2*, the* 0*-region* $\{\mathtt{e}\}$ *is closed in the corresponding subgame. However, it is not closed in the whole game, because of the move leading to* $\mathtt{c}$*,* i.e.*, to the region of measure* 4*. Proposition 7.2.2 can now be applied and a promotion of* $\{\mathtt{e}\}$ *to* 4 *is performed, resulting in the new* 0*-region* $\{\mathtt{c},\mathtt{e},\mathtt{j}\}$ *that resets* 1*-region* $\{\mathtt{d}\}$*. The search resumes at the corresponding priority and, after computing the extension of such a region via the attractor, we obtain that it is still open in the corresponding subgame. Consequently, the* 1*-region* $\{\mathtt{d}\}$ *of measure* 3 *is recomputed and, then, priority* 1 *is processed to build the* 1*-region* $\{\mathtt{g}\}$*. The latter is closed in the associated subgame, but not in the original game, because of a move leading to position* $\mathtt{d}$*. Hence, another promotion is performed, leading to the closed region of measure 3 at Column 3, which in turn triggers a promotion to* 5*. When the promotion of* 0*-region* $\{\mathtt{i}\}$ *to priority* 6 *is performed, however,* 0*-region* $\{\mathtt{c},\mathtt{e},\mathtt{j}\}$ *of measure*

4 is not reset. This leads directly to the configuration in Column 6, after the maximization of 0*-region* 6*, which attracts* b*,* d*,* g*, and* j*. Notice that, as prescribed by Proposition 7.2.3, the set* {c, e}*, highlighted by the gray area, is still a* 0*-region. On the other hand, the set* {f, h}*, highlighted by the dashed line and originally included in* 1*-region* {b, d, f, g, h} *of priority* 5*, needs to be reset, since it is not a* 1*-region any more. It is, actually, an open* 0*-region instead. Now,* 0*-region* 4 *is closed in its subgame and it is promoted to* 6*. As result of this promotion, we obtain the closed* 0*-region* {a, b, c, d, e, g, i, j}*, which is a dominion for player* 0*.*

We can now provide the formal account of the PP+ dominion space. We shall denote with Rg the set of region pairs in $\mathfrak{D}$ and with Rg^- and Rg^+ the sets of open and closed region pairs, respectively.

Similarly to the priority promotion algorithm, during the search for a dominion, the computed regions, together with their current measure, are kept track of by means of an auxiliary priority function $\mathsf{r} \in \Delta \triangleq \mathrm{Ps} \to \mathrm{Pr}$, called *region function*. Given a priority $p \in \mathrm{Pr}$, we denote by $\mathsf{r}^{(\geq p)}$ (*resp.*, $\mathsf{r}^{(>p)}$, $\mathsf{r}^{(<p)}$, and $\mathsf{r}^{(\equiv_2 p)}$) the function obtained by restricting the domain of r to the positions with measure greater than or equal to p (*resp.*, greater than, lower than, and congruent modulo 2 to p). Formally, $\mathsf{r}^{(\sim p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathrm{Ps} : \mathsf{r}(v) \sim p\}$, for $\sim \in \{\geq, >, <, \equiv_2\}$. By $\mathfrak{D}_{\mathsf{r}}^{\leq p} \triangleq \mathfrak{D} \setminus \mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$, we denote the largest subgame obtained by removing from $\mathfrak{D}$ all the positions in the domain of $\mathsf{r}^{(>p)}$. The *maximization* of a priority function $\mathsf{r} \in \Delta$ is the unique priority function $\mathsf{m} \in \Delta$ such that $\mathsf{m}^{-1}(q) = \mathsf{atr}^{\wp}_{\mathfrak{D}^{\leq q}_{\mathsf{m}}}(\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}^{\leq q}_{\mathsf{m}}})$, for all priorities $q \in \mathsf{rng}(\mathsf{r})$ with $\wp \triangleq q \bmod 2$. In addition, we say that r is *maximal* above $p \in \mathrm{Pr}$ iff $\mathsf{r}^{(>p)} = \mathsf{m}^{(>p)}$.

As opposed to the PP approach, where a promotion to $p \equiv_2 \wp$ resets all the regions lower than p, here we need to take into account the fact that the regions of the opponent $\overline{\wp}$ are reset, while the ones of player $\wp$ are retained. In particular, we need to ensure that, as the search proceeds from p downward to any priority $q < p$, the maximization of the regions contained at priorities higher than q can never make the region recorded in r at q invalid. To this end, we consider only priority functions r that satisfy the requirement that, at all priorities, they contain regions *w.r.t.* the subgames induced by their maximizations m. Formally, $\mathsf{r} \in \mathbb{R} \subseteq \Delta$ is a *region function* iff, for all priorities $q \in \mathsf{rng}(\mathsf{m})$ with $\wp \triangleq q \bmod 2$, it holds that $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathfrak{D}^{\leq q}_{\mathsf{m}}}$ is an $\wp$-region in the subgame $\mathfrak{D}^{\leq q}_{\mathsf{m}}$, where m is the maximization of r.

The status of the search of a dominion is encoded by the notion of *state* s of the dominion space, which contains the current region function r and the current priority p reached by the search in $\mathfrak{D}$. Initially, r coincides with the priority function pr of the entire game $\mathfrak{D}$, while p is set to the maximal priority $\mathsf{pr}(\mathfrak{D})$ available in the game. To each of such states $s \triangleq (\mathsf{r}, p)$, we then associate the *subgame at* s

defined as $\partial_s \triangleq \partial_{\mathsf{r}}^{\leq p}$, representing the portion of the original game that still has to be processed.

The following state space specifies the configurations in which the PP+ procedure can reside and the relative order that the successor function must satisfy.

Definition 7.2.4 (State Space for PP+)**.** *A* PP+ state space *is a tuple* $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$*, where:*

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

Condition 1 requires that every region $\mathsf{r}^{-1}(q)$ with measure $q > p$ be $\wp$-maximal, where $\wp = q \bmod 2$. This implies that $\mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\partial_{\mathsf{r}}^{\leq q}}$. Moreover, the current priority p of the state must be one of the measures recorded in r. In addition, Condition 2 specifies the initial state, while Condition 3 defines the ordering relation among states, which the successor operation has to comply with. It asserts that a state s_1 is strictly smaller than another state s_2 if either there is a region recorded in s_1 with some higher measure q that strictly contains the corresponding one in s_2 and all regions with measure grater than q are equal in the two states, or state s_1 is currently processing a lower priority than the one of s_2.

A region pair $(\mathrm{R}, \wp)$ is compatible with a state $s \triangleq (\mathsf{r}, p)$ if it is an $\wp$-region in the current subgame ∂_s. Moreover, if such a region is $\wp$-open in that game, it has to be $\wp$-maximal and needs to necessarily contain the current region $\mathsf{r}^{-1}(p)$ of priority p in r.

Definition 7.2.5 (Compatibility Relation)**.** *An open quasi dominion pair* $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ *is* compatible *with a state* $s \triangleq (\mathsf{r}, p) \in \mathrm{S}$*, in symbols* $s \succ (\mathrm{R}, \wp)$*, iff* (1) $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\partial_s}$ *and* (2) *if* R *is* $\wp$*-open in* ∂_s *then* $\mathrm{R} = \mathsf{atr}_{\partial_s}^{\wp}(\mathsf{r}^{-1}(p))$.

Algorithm 13: Query Function.

```
signature ℜ : S → 2^Ps × {0,1}
function ℜ(s)
    let (r, p) = s in
1       ℘ ← p mod 2
2       R ← atr^℘_{∂_s}[r^{-1}(p)]
3   return (R, ℘)
```

Algorithm 13 provides the implementation of the query function compatible with the priority-promotion mechanism. Line 1 simply computes the parity $\wp$ of the priority to process in the state $s \triangleq (\mathsf{r}, p)$. Line 2, instead, computes the attractor *w.r.t.* player $\wp$ in subgame ∂_s of the region contained in r at the current priority p. The resulting set R is, according to Proposition 7.2.1, an $\wp$-maximal $\wp$-region of ∂_s containing $\mathsf{r}^{-1}(p)$.

The promotion operation is based on the notion of best escape priority mentioned above, namely the priority of the lowest $\wp$-region in r that has an incoming move coming from the $\wp$-region, closed in the current subgame, that needs to be promoted. This concept is formally defined as follows. Let $I \triangleq Mv \cap ((\mathrm{R} \cap \mathrm{Ps}^{\overline{\wp}}) \times (\mathsf{dom}(\mathsf{r}) \setminus \mathrm{R}))$ be the *interface relation* between R and r, *i.e.*, the set of $\overline{\wp}$-moves exiting from R and reaching some position within a region recorded in r. Then, $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r})$ is set to the minimal measure of those regions that contain positions reachable by a move in I. Formally, $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r}) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r} \restriction \mathsf{rng}(I)))$. Such a value represents the best priority associated with an $\wp$-region contained in r and reachable by $\overline{\wp}$ when escaping from R. Note that, if R is a closed $\wp$-region in $\ni_s$, then $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r})$ is necessarily of parity $\wp$ and greater than the measure p of R. This property immediately follows from the maximality of r above p. Indeed, no move of an $\overline{\wp}$-position can lead to a $\overline{\wp}$-maximal $\overline{\wp}$-region. For instance, for 0-region $\mathrm{R} = \{\mathsf{e}\}$ with measure 2 in Column 1 of Figure 7.1, we have that $I = \{(\mathsf{e}, \mathsf{a}), (\mathsf{e}, \mathsf{c})\}$ and $\mathsf{r} \restriction \mathsf{rng}(I) = \{(\mathsf{a}, 6), (\mathsf{c}, 4)\}$. Hence, $\mathsf{bep}^{1}(\mathrm{R}, \mathsf{r}) = 4$.

Algorithm 14 reports the pseudo-code of the successor function, which differs from the one proposed in [BDM18b] only in Line 5, where Proposition 7.2.3 is applied. Given the current state s and a compatible region pair $(\mathrm{R}, \wp)$ open in the whole game as inputs, it produces a successor state $s^\star \triangleq (\mathsf{r}^\star, p^\star)$ in the dominion space. It first checks whether R is open also in the subgame $\ni_s$ (Line 1). If this is the case, it assigns the measure p to region R and stores it in the new region function $\mathsf{r}^\star$ (Line 2). The new current priority $p^\star$ is, then, computed as the highest priority lower than p in $\mathsf{r}^\star$ (Line 3). If, on the other hand, R is closed in $\ni_s$, a promotion, merging R with some other $\wp$-region contained in r, is required. The next priority $p^\star$ is set to the bep of R for player $\overline{\wp}$ in the entire game $\ni$ *w.r.t.* r (Line 4). Region R is, then, promoted to priority $p^\star$ and all and only the regions of the opponent $\overline{\wp}$ with lower measure than $p^\star$ in the region function r are reset by means of the completion operator defined in Chapter 2 (Line 5).

Algorithm 14: Successor Function.

```
   signature ↓ : ≻ → Δ × Pr
   function s↓(R, ℘)
       let (r, p) = s in
1          if (R, ℘) ∈ Rg⁻_{∋_s} then
2              r* ← r[R ↦ p]
3              p* ← max(rng(r*^(<p)))
           else
4              p* ← bep^{℘̄}[R, r]
5              r* ← pr ⊎ r^{(≥p*)∨(≡₂p*)}[R ↦ p*]
6      return (r*, p*)
```

The following theorem asserts that the PP+ state space, together with the same query function of PP and the successor function of Algorithm 14 is a dominion space.

Theorem 7.2.1 (PP+ Dominion Space)**.** *For a game ∂, the* PP+ structure $\mathcal{D} \triangleq \langle \partial, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where $\mathcal{S}$ is given in Definition 7.2.4, $\succ$ is the relation of Definition 7.2.5, and $\Re$ and $\downarrow$ are the functions computed by Algorithms 13 and 14 is a dominion space.*

To prove the above theorem, we have to show that the three components $\mathcal{S}$, $\Re$, and $\downarrow$ of the structure $\mathcal{D}$ satisfy the properties required by Definition 7.2.2 of dominion space. We do this through the Lemmas 7.2.1, 7.2.2, and 7.2.3.

Lemma 7.2.1 (State Space)**.** *The* PP+ *state space $\mathcal{S} = \langle \mathrm{S}, \top, \prec \rangle$ for a game $\partial \in \mathcal{P}$ is a well-founded partial order* w.r.t. $\prec$ *with designated element $\top \in \mathrm{S}$.*

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

For the irreflexive property, by Item 3 of Definition 7.2.4, it is immediate to see that $s \not\prec s$, for all states $s \triangleq (\mathsf{r}, p) \in \mathrm{S}$, since neither there exists a priority $q \in \mathsf{rng}(\mathsf{r})$ such that $\mathsf{r}^{-1}(q) \subset \mathsf{r}^{-1}(q)$ nor $p < p$.

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

- Item 3.a for both $s_1 \prec s_2$ and $s_2 \prec s_3$: there exist two priorities $q_1 \in \mathsf{rng}(\mathsf{r}_1)$ and $q_2 \in \mathsf{rng}(\mathsf{r}_2)$ with $q_1 \geq p_1$ such that ${\mathsf{r}_1}^{(>q_1)} = {\mathsf{r}_2}^{(>q_1)}$, ${\mathsf{r}_2}^{(>q_2)} = {\mathsf{r}_3}^{(>q_2)}$, $\mathsf{r}_2^{-1}(q_1) \subset \mathsf{r}_1^{-1}(q_1)$, and $\mathsf{r}_3^{-1}(q_2) \subset \mathsf{r}_2^{-1}(q_2)$. Let $q \triangleq \mathsf{max}\{q_1, q_2\} \geq p_1$. If $q = q_1 = q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. If $q = q_1 > q_2$ then ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = ({\mathsf{r}_2}^{(>q_2)})^{(>q)} = ({\mathsf{r}_3}^{(>q_2)})^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Finally, if $q = q_2 > q_1$ then ${\mathsf{r}_1}^{(>q)} = ({\mathsf{r}_1}^{(>q_1)})^{(>q)} = ({\mathsf{r}_2}^{(>q_1)})^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$. Moreover, $q \in \mathsf{rng}\big({\mathsf{r}_2}^{(>q_1)}\big) = \mathsf{rng}\big({\mathsf{r}_1}^{(>q_1)}\big) \subseteq \mathsf{rng}(\mathsf{r}_1)$. Summing up, it holds that $s_1 \prec s_3$.

- Item 3.a for $s_1 \prec s_2$ and Item 3.b for $s_2 \prec s_3$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_1)$ with $q \geq p_1$ such that ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)}$ and $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$; moreover, $\mathsf{r}_2 = \mathsf{r}_3$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) = \mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$. Consequently, $s_1 \prec s_3$.

- Item 3.a for $s_2 \prec s_3$ and Item 3.b for $s_1 \prec s_2$: there exists a priority $q \in \mathsf{rng}(\mathsf{r}_2)$ with $q \geq p_2$ such that ${\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$ and $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q)$; moreover, $\mathsf{r}_1 = \mathsf{r}_2$ and $p_1 < p_2$. Thus, ${\mathsf{r}_1}^{(>q)} = {\mathsf{r}_2}^{(>q)} = {\mathsf{r}_3}^{(>q)}$, $\mathsf{r}_3^{-1}(q) \subset \mathsf{r}_2^{-1}(q) = \mathsf{r}_1^{-1}(q)$, $q \in \mathsf{rng}(\mathsf{r}_1)$, and $q > p_1$. Consequently, $s_1 \prec s_3$.

- Item 3.b for both $s_1 \prec s_2$ and $s_2 \prec s_3$: $\mathsf{r}_1 = \mathsf{r}_2$, $\mathsf{r}_2 = \mathsf{r}_3$, $p_1 < p_2$, and $p_2 < p_3$. Hence, $\mathsf{r}_1 = \mathsf{r}_3$ and $p_1 < p_3$, which implies that $s_1 \prec s_3$ also in this case.

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

Lemma 7.2.2 (Query Function)**.** *The function $\Re$ is a query function,* i.e.*, for all states $s \in \mathrm{S}$, it holds that* (1) $\Re(s) \in \mathrm{QD}$ *and* (2) *if* $\Re(s) \in \mathrm{QD}^{-}$ *then* $s \succ \Re(s)$.

Proof. Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}$ be a state and $(\mathrm{R}, \wp) \triangleq \Re(s)$ the pair consisting of the set of positions $\mathrm{R} \subseteq \mathrm{Ps}$ and the player $\wp \in \{0, 1\}$ returned by the function $\Re$ on input s. Due to Line 1 of Algorithm 13, it follows that $\wp \equiv_2 p$. By Item 1 of Definition 7.2.4, we have that $p \in \mathsf{rng}(\mathsf{r})$, so $\mathsf{r}^{-1}(p) \neq \emptyset$, and $\mathsf{r}^{-1}(p) \subseteq \mathrm{Ps}_{\mathcal{D}_s}$. Thus, by definition of region function, it holds that $\mathsf{r}^{-1}(p)$ is an $\wp$-region in $\mathcal{D}_s$. Now, by Line 2 of Algorithm 13 and Proposition 7.2.1, we obtain that $\mathrm{R} = \mathsf{atr}^{\wp}_{\mathcal{D}_s}([\mathsf{r}^{-1}(p)) \subseteq \mathrm{Ps}_{\mathcal{D}_s}$ is an $\wp$-region in $\mathcal{D}_s$, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\mathcal{D}_s}$, and, so a quasi dominion in $\mathcal{D}$, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{QD}$. In addition, $s \succ (\mathrm{R}, \wp)$, since all requirements of Definition 7.2.5 are satisfied. □

Lemma 7.2.3 (Successor Function)**.** *The function $\downarrow$ is a successor function,* i.e.*, for all states $s \in \mathrm{S}$ and quasi dominion pairs $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ with $s \succ (\mathrm{R}, \wp)$, it holds that* (1) $s{\downarrow}(\mathrm{R}, \wp) \in \mathrm{S}$ *and* (2) $s{\downarrow}(\mathrm{R}, \wp) \prec s$.

Proof. Let $s \triangleq (\mathsf{r}, p) \in \mathrm{S}$ be a state, $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ an open quasi dominion pair in $\mathcal{D}$ compatible with s, and $s^{\star} = (\mathsf{r}^{\star}, p^{\star}) \triangleq s{\downarrow}(\mathrm{R}, \wp)$ the result obtained by computing the function $\downarrow$ on s and $(\mathrm{R}, \wp)$. Due to Item 1.a of Definition 7.2.4, we have that *(1)* r is maximal above p. Moreover, by Item 1 of Definition 7.2.5, it holds that $\mathrm{R} \subseteq \mathrm{Ps}_{\mathcal{D}_s}$, which implies *(2)* $\mathsf{dom}\big(\mathsf{r}^{(>p)}\big) \cap \mathrm{R} = \emptyset$.

On the one hand, suppose that R is $\wp$-open in $\mathcal{D}_s$, *i.e.*, $(\mathrm{R}, \wp) \in \mathrm{Rg}^{-}_{\mathcal{D}_s}$. By Lines 1-3 of Algorithm 14, we have that *(3)* $\mathsf{r}^{\star} = \mathsf{r}[\mathrm{R} \mapsto p]$ and *(4)* $p^{\star} = \mathsf{max}(\mathsf{rng}\Big(\mathsf{r}^{\star(<p)}\Big))$. In addition, by Item 2 of Definition 7.2.5, it holds that *(5)* $\mathrm{R} = \mathsf{atr}^{\wp}_{\mathcal{D}_s}(\mathsf{r}^{-1}(p))$. Now, by Point *(4)*, it immediately follows that *(6)* $p^{\star} \in \mathsf{rng}(\mathsf{r}^{\star})$, *(7)* $p^{\star} < p$, and *(8)* there is no priority $q \in \mathsf{rng}(\mathsf{r}^{\star})$ such that $p^{\star} < q < p$. Also, by Points *(2)*, *(3)* and *(5)*, we have that *(9)* $\mathsf{r}^{\star(>p)} = \mathsf{r}^{(>p)}$, *(10)* $\mathsf{r}^{\star -1}(p) = \mathrm{R}$, and *(11)* $\mathsf{r}^{\star(<p)} = \mathsf{r}^{(<p)}{\restriction}(\mathrm{Ps} \setminus \mathrm{R})$. Thus, by Points *(1)*, *(5)*, *(8)*, *(9)*, and *(10)*, it holds that *(12)* $\mathsf{r}^{\star}$ is maximal above $p^{\star}$. Finally, to prove that *(13)* $\mathsf{r}^{\star}$ is a region function, we need to show that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^{\star}}}$ is a β-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}^{\star}}$, for all priorities $q \in \mathsf{rng}(\mathsf{m}^{\star})$ with $\beta \triangleq q \bmod 2$, where $\mathsf{m}^{\star}$ is the maximisation of $\mathsf{r}^{\star}$. To this aim, we observe

that, by definition of state space, r is a region function, so, the above property is surely satisfied by r *w.r.t.* its maximisation m.

- If $q > p$, by Points *(1)* and *(9)*, we have that $\mathsf{r}^{\star -1}(q) = \mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}} = \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}}$. Therefore, the thesis immediately follows, since $\mathsf{r}^{-1}(q)$ is a β-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}}$, where $\beta \triangleq q \bmod 2$.

- If $q = p$, first observe that $\mathcal{D}_s = \mathcal{D}^{\leq q}_{\mathsf{m}} = \mathcal{D}^{\leq q}_{\mathsf{m}^\star}$. By Points *(2)*, *(5)*, and *(10)*, we have that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}} = \mathrm{R}$. So, the property is ensured by the fact that R is an $\wp$-region in $\mathcal{D}_s$.

- For the last case $q < p$, notice that $\mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}} \cap \mathrm{R} = \emptyset$. Therefore, by Point *(11)*, $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}} = \mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}}$. Hence, the thesis follows, since $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}}$ is a β-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}}$, where $\beta \triangleq q \bmod 2$.

Summing up, Points *(13)*, *(12)*, and *(6)* ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}$. At this point, it remains just to show that $(\mathsf{r}^\star, p^\star) \prec (\mathsf{r}, p)$. By Point *(5)*, two cases may arise. If $\mathsf{r}^{-1}(p) \subset \mathrm{R}$, due to Points *(9)* and *(10)*, the thesis follows from Item 3.a of Definition 7.2.4, where the priority q is set to p. On the contrary, if $\mathrm{R} = \mathsf{r}^{-1}(p)$, by Point *(3)*, we have that $\mathsf{r}^\star = \mathsf{r}$. Therefore, due to Point *(7)*, the thesis is derived from Item 3.b of the same definition.

On the other hand, suppose that R is $\wp$-closed in $\mathcal{D}_s$, *i.e.*, $(\mathrm{R}, \wp) \notin \mathrm{Rg}^{-}_{\mathcal{D}_s}$. By Lines 1, 4, and 5 of Algorithm 14, we have that *(14)* $p^\star = \mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r})$ and *(15)* $\mathsf{r}^\star = \mathsf{pr} \uplus \mathsf{r}^{(\geq p^\star) \vee (\equiv_2 p^\star)}[\mathrm{R} \mapsto p^\star]$. By Points *(2)* and *(14)*, the definition of bep, and the fact that R is closed, it is not hard to see that *(16)* $p^\star > p$. Now, by Points *(15)* and *(2)*, it follows that *(17)* $\mathsf{r}^{\star (> p^\star)} = \mathsf{r}^{(> p^\star)}$, *(18)* $\mathsf{r}^{\star -1}(p^\star) = \mathsf{r}^{-1}(p^\star) \cup \mathrm{R}$, *(19)* $\mathsf{r}^{\star (< p^\star) \wedge (\not\equiv_2 p^\star)} \subseteq \mathsf{pr}^{(< p^\star) \wedge (\not\equiv_2 p^\star)}$. Hence, by Points *(1)*, *(16)*, and *(17)*, it holds that *(20)* $\mathsf{r}^\star$ is maximal above $p^\star$. Moreover, by Point *(18)*, we have that *(21)* $p^\star \in \mathsf{rng}(\mathsf{r}^\star)$. Finally, we have to prove that *(22)* $\mathsf{r}^\star$ is a region function, *i.e.*, $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}}$ is a β-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}^\star}$, for all priorities $q \in \mathsf{rng}(\mathsf{m}^\star)$ with $\beta \triangleq q \bmod 2$, where $\mathsf{m}^\star$ is the maximisation of $\mathsf{r}^\star$.

- If $q > p^\star$, by Points *(1)*, *(16)* and *(17)*, we have that $\mathsf{r}^{\star -1}(q) = \mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}} = \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}}$. Therefore, the thesis immediately follows, since $\mathsf{r}^{-1}(q)$ is a β-region in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}}$, where $\beta \triangleq q \bmod 2$.

- If $q = p^\star$, by Points *(1)*, *(2)*, *(16)* and *(18)*, we have that $\mathsf{r}^{\star -1}(q) \subseteq \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}}}$. Now, by Point *(1)* and the fact that r is a region function, it follows that $\mathsf{r}^{-1}(q)$ is an $\wp$-region in $\mathcal{D}^{\leq q}_{\mathsf{m}} = \mathcal{D}^{\leq q}_{\mathsf{m}^\star}$. Moreover, due to Point *(14)*, R is an $\wp$-dominion in $\mathcal{D}^{\leq q}_{\mathsf{m}} \setminus \mathsf{r}^{-1}(q)$. Therefore, due to Proposition 7.2.2, it holds that $\mathsf{r}^{\star -1}(q)$ is an $\wp$-region in $\mathcal{D}^{\leq q}_{\mathsf{m}^\star}$.

- If $q < p^\star$ and $q \not\equiv_2 p^\star$, by Point *(19)*, we immediately have that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\mathcal{D}^{\leq q}_{\mathsf{m}^\star}}$ is a $\overline{\wp}$-region in $\mathcal{D}^{\leq q}_{\mathsf{m}^\star}$, since it only contains positions of priority q, which is the maximal one in the subgame $\mathcal{D}^{\leq q}_{\mathsf{m}^\star}$.

- If $q < p^\star$ and $q \equiv_2 p^\star$, it can be easily shown that $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{m}^\star}^{\leq q}} = (\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{m}}^{\leq q}} \setminus \mathrm{R}^\star) \cup \mathrm{P}$, for some $\wp$-maximal $\wp$-region $\mathrm{R}^\star$ in ∂_s and some set of positions $\mathrm{P} \subseteq \mathsf{pr}^{-1}(q)$ of priority q. Now, by applying Proposition 7.2.3, we have that $\mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{m}}^{\leq q}}$ is an $\wp$-region in $\partial_{\mathsf{m}}^{\leq q}$. Therefore, also $\mathsf{r}^{\star -1}(q) \cap \mathrm{Ps}_{\partial_{\mathsf{m}^\star}^{\leq q}}$ is an $\wp$-region in $\partial_{\mathsf{m}^\star}^{\leq q}$, since it only contains further positions of maximal priority q in the corresponding subgame.

Summing up, Points *(20)*, *(21)*, and *(22)* ensure that $(\mathsf{r}^\star, p^\star) \in \mathrm{S}$. At this point, as for the previous case, it remains to show that $(\mathsf{r}^\star, p^\star) \prec (\mathsf{r}, p)$. This fact easily follows from Item 3.a of Definition 7.2.4, where the priority q is set to $p^\star$, since, by Points *(2)* and *(18)*, we have that $\mathsf{r}^{-1}(p^\star) \cap \mathrm{R} = \emptyset$, so $\mathsf{r}^{-1}(p^\star) \subset \mathsf{r}^{-1}(p^\star) \cup \mathrm{R} = \mathsf{r}^{\star -1}(p^\star)$. □

The PP+ procedure does reduce, *w.r.t.* PP, the number of reset needed to solve a game and the exponential worst-case game presented in [BDM18b] does not work any more. However, a worst-case, which is a slight modification of the one for PP, does exist for this procedure as well.

Consider the game $\partial_h^{\mathrm{PP}+}$ containing h chains of length 2 that converge into a single position of priority 0 with a self loop. The i-th chain has a head of priority $2(h+1) - i$ and a body composed of a single position with priority i and a self loop. An instance of this game with $h = 4$ is depicted in Figure 7.2. The labels of the positions correspond to the associated priorities. Intuitively, the execution depth of the PP+ dominion space for this game is exponential, since the consecutive promotion operations performed on each chain can simulate the increments of a partial form of binary counter, some of whose configurations are missing. As a result, the number of configurations of the counter follows a Fibonacci-like sequence of the form $F(h) = F(h-1) + F(h-2) + 1$, with $F(0) = 1$ and $F(1) = 2$. It is interesting to note that this is the same recurrence equation of the number of minimal AVL trees of hight h. The search procedure on $\partial_4^{\mathrm{PP}+}$ starts by building the following four open regions: the 1-region $\{9\}$, the 0-region $\{8\}$, the 1-region $\{7\}$, and 0-region $\{6\}$. This state represents the configuration of the counter, where all four digits are set to 0. The closed 0-region $\{4\}$ is then found and promoted to 6. Now, the counter is set to 0001. After that, the closed 1-region $\{3\}$ is computed that is promoted to 7. Due to the promotion to 7, the positions in the 0-region with priority 6 are reset to their original priority, as they belong to the opponent player. This releases the chain with head 6, which corresponds to the reset of the least significant digit of the counter caused by the increment of the second one, *i.e.*, the counter displays 0010. The search resumes at priority 6 and the 0-regions $\{6\}$ and

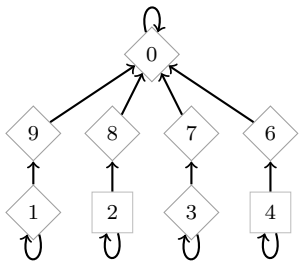


Figure 7.2: The $\partial_4^{\mathrm{PP}+}$ game.

$\{4\}$ are computed once again. A second promotion of $\{4\}$ to 6 is performed, resulting in the counter assuming value 0011. When the closed 0-region $\{2\}$ is promoted to 8, however, only the 1-region $\{7,3\}$ is reset, leading to configuration 0101 of the counter. Hence, configuration 0100 is skipped. Similarly, when, the counter reaches configuration 0111 and 1-region $\{1\}$ is promoted to 9, the 0-regions $\{8,2\}$ and $\{6,4\}$ are reset, leaving 1-region $\{7,3\}$ intact. This leads directly to configuration 1010 of the counter, skipping configurations 1000 and 1001.

An estimate of the depth of the PP+ dominion space on the game $\eth_h^{\mathrm{PP}+}$ is given by the following theorem.

Theorem 7.2.2 (Execution-Depth Lower Bound)**.** *For all $h \in \mathbb{N}$, there exists a* PP+ *dominion space $\mathcal{D}_h^{\mathrm{PP}+}$ with $k = 2h+1$ positions and priorities, whose execution depth is $\mathsf{Fib}(2(h+4))/\mathsf{Fib}(h+4)-(h+6) = \Theta\big(((1+\sqrt{5})/2)^{k/2}\big)$.*

Proof. The game $\eth_h^{\mathrm{PP}+}$ encodes a partial binary counter, where each one of the h chains, composed of two positions connected to position 0, represents a digit. The promotion operation of region i to region $2(h+1)-i$ corresponds to the increment of the i-th digit. As a consequence, the number of promotions for the PP+ algorithm is equal to the number of configurations of the counter, except for the initial one. To count them all on game $\eth_h^{\mathrm{PP}+}$, consider the recursive function $\mathsf{P}(h)$ having base case $\mathsf{P}(0) = 1$ and inductive cases $\mathsf{P}(h) = 1 + \mathsf{P}(h-1) + \mathsf{G}(h-1)$, where the function G is described below. To be precise, this function computes a value that is exactly one more than the number of promotions. The correctness of the base case is trivial. Indeed, when h is equal to 0, there are no chains and, so, no promotions. We can now focus on the inductive case $h > 0$. Before reaching the promotion of region $\{1\}$ to region $\{2h+1\}$, *i.e.*, to set the most-significant digit, all others digits must be set. To do this, the amount of necessary promotions is equal to those required by the game with $h-1$ chains, *i.e.*, $\mathsf{P}(h-1)$. At this point, an additional promotion is counted when region $\{1\}$ is merged to region $\{2h+1\}$. As shown in the execution of the game depicted in Figure 7.2, this promotion resets the digits with index $i < h$ such that $i \not\equiv_2 h$. Therefore, $\mathsf{G}(h-1)$ counts exactly the promotions needed to set to 1 the digits reset after this promotion.

We now prove that the function G can be expressed by means of the following recursive definition: $\mathsf{G}(0) = 0$ and $\mathsf{G}(h) = \lceil h/2 \rceil + \sum_{i=0}^{\lceil h/2 \rceil - 1} \mathsf{G}(2i + 1 - h \bmod 2)$. When $h = 0$, we have that $\mathsf{G}(h) = 0$, since no chain is present. For the inductive case, when $h > 0$, if h is odd, the least significant digit is not set. Therefore, the algorithm performs a promotion to set it. After this digit is set, $\mathsf{G}(0) = 0$ promotions are required, since no lower chain is present. Then, the algorithm reaches the third least significant digit, since the second one is already set. In order to set it, another promotion is required.

Once the third digit is set, the second one gets reset. So, to set all digits, $\mathsf{G}(2)$ promotions are then required. After this, the algorithm will reach the fifth significant digit and proceeds similarly until the $(h-1)$-th digit is processed. Summing up, $(h+1)/2 + \sum_{i=0}^{(h-1)/2} \mathsf{G}(2i)$ promotions are required in total. Similarly, when h is even, we obtain that $\mathsf{G}(h) = (h/2) + \sum_{i=0}^{(h/2)-1} \mathsf{G}(2i+1)$. Consequently $\mathsf{G}(h) = \lceil h/2 \rceil + \sum_{i=0}^{\lceil h/2 \rceil - 1} \mathsf{G}(2i+1-h \bmod 2)$, regardless of the parity of h.

Finally, it can be proved by induction that $\mathsf{G}(h) = \mathsf{Fib}(h+2)-1$, and, consequently, $\mathsf{P}(h) = \mathsf{Fib}(h+3)-1$. The proof of these claim requires the application of the equalities $\mathsf{Fib}(2h) = \sum_{i=0}^{h-1} \mathsf{Fib}(2i+1)$, and $\mathsf{Fib}(2h+1) - 1 = \sum_{i=1}^{h} \mathsf{Fib}(2i)$. Observe that the above definition of P is equivalent to the sequence for the number of configuration claimed for $\mathfrak{I}_h$, *i.e.* $\mathsf{P}(h) = \mathsf{P}(h-1) + \mathsf{P}(h-2) + 1$, since it is easy to see that $\mathsf{G}(h-1) = \mathsf{P}(h-2)$.

A similar reasoning can be exploited to count the number of queries executed by the the PP+ algorithm on game $\mathfrak{I}_h^{\mathrm{PP}+}$. To do this, we use the function Q, having base case $\mathsf{Q}(0) = 1$ and inductive case $\mathsf{Q}(h) = 3 + \mathsf{Q}(h-1) + \mathsf{H}(h-1)$, where $\mathsf{H}(h)$ is defined in the following. The base case is trivial. When h is equal to 0, there are no chains and, so, the only region that can be returned by the query function is $\{0\}$. Let us now focus on the inductive case $h > 0$. The algorithm requires one query to create region $\{2h+1\}$, corresponding to the head of the h-th chain. Before reaching the construction of region $\{1\}$, all remaining digits have to be set to 1. To do this, the amount of necessary queries is equal to $\mathsf{Q}(h-1)$, where the query of region $\{0\}$ actually correspond to the query of region $\{1\}$. At this point, one query is needed to promote the body $\{1\}$ to the head $\{2h+1\}$ of the h-th chain. This promotion resets the digits with index $i < h$ such that $i \not\equiv_2 h$. Therefore, $\mathsf{H}(h-1)$ queries are needed to set to 1 the digits reset after it. Finally, one query is required to create region $\{0\}$.

We now prove that the function H can be expressed by the following recursive definition: $\mathsf{H}(0) = 0$ and $\mathsf{H}(h) = h + 2\lceil h/2 \rceil + \sum_{i=0}^{\lceil h/2 \rceil - 1} \mathsf{H}(2i+1-h \bmod 2)$. Indeed, $\mathsf{H}(h) = 0$, when $h = 0$, since no chain is present. For the inductive case, when $h > 0$, the function requires h queries to create the regions corresponding to the heads of the chains before reaching the first region to promote. Also, it requires two queries for each of the $\lceil h/2 \rceil$ promotions that reset the chains of lower significance.

By induction, it can be proved that $\mathsf{H}(h) = \mathsf{Luc}(h+3)-4$, and, consequently, $\mathsf{Q}(h) = \mathsf{Luc}(h+4)-h-6$, where $\mathsf{Luc}(h)$ is the h-th Lucas number with base cases $\mathsf{Luc}(0) = 2$ and $\mathsf{Luc}(1) = 1$. The proof of these claim requires the application of the equalities $\mathsf{Luc}(2h+1) + 1 = \sum_{i=0}^{h} \mathsf{Luc}(2i)$, and $\mathsf{Luc}(2h) - 2 = \sum_{i=0}^{h-1} \mathsf{Luc}(2i+1)$.

In conclusion, since it is known that $\mathsf{Luc}(h) = \mathsf{Fib}(2h)/\mathsf{Fib}(h)$, the function $\mathsf{Q}(h)$ can be expressed as $\mathsf{Fib}(2(h+4)/\mathsf{Fib}(h+4) - (h+6)$, whose asymptotic behavior is a $\Theta\big(((1+\sqrt{5})/2)^{k/2}\big)$, as claimed in the theorem. □

7.3 Delayed Promotion Policy

At the beginning of the previous section, we have observed that the time complexity of the dominion search procedure $\mathsf{src}_{\mathcal{D}}$ linearly depends on the execution depth of the underlying dominion space $\mathcal{D}$. This, in turn, depends on the number of promotions performed by the associated successor function and is tightly connected with the reset mechanism applied to the regions with measure lower than the one of the target region of the promotion. In fact, it can be proved that, when no resets are performed, the number of possible promotions is bounded by a polynomial in the number of priorities and positions of the game under analysis. Consequently, the exponential behaviors exhibited by the PP algorithm and its enhanced version PP+ are strictly connected with the particular reset mechanism employed to ensure the soundness of the promotion approach. The correctness of the PP+ method shows that the reset can be restricted to only the regions of opposite parity *w.r.t.* the one of the promotion and, as we shall show in the next section, this enhancement is also relatively effective in practice. However, we have already noticed that this improvement does not suffice to avoid some pathological cases and no general finer criteria is available to avoid the reset of the opponent regions. Therefore, to further reduce such resets, in this section we propose a finer promotion policy that tries to reduce the application of the reset mechanism. The new solution procedure is based on delaying the promotions of regions, called *locked promotions*, that require the reset of previously performed promotions of the opponent parity, until a complete knowledge of the current search phase is reached. Once only locked promotions are left, the search phase terminates by choosing the highest measure $p^\star$ among those associated with the locked promotions and, then, performing all the postponed ones of the same parity as $p^\star$ altogether. In order to distinguish between locked and unlocked promotions, the corresponding target priorities of the performed ones, called *instant promotions*, are recorded in a supplementary set P. Moreover, to keep track of the locked promotions, a supplementary partial priority function $\widetilde{\mathsf{r}}$ is used. In more detail, the new procedure evolves exactly as the PP+ algorithm, as long as open regions are discovered. When a closed one with measure p is provided by the query function, two cases may arise. If the corresponding promotion is not locked, the destination priority q is recorded in the set P and the instant promotion is performed similarly to the case of PP+. Otherwise, the promotion is not performed. Instead, it is recorded in the supplementary function $\widetilde{\mathsf{r}}$, by assigning to R in $\widetilde{\mathsf{r}}$ the target priority q of that promotion and in r its current measure p. Then, the positions in R are removed from the subgame and the search proceeds at the highest remaining priority, as in the case R was open in the subgame. In case the region R covers the entire subgame, all priorities available in the original game have been processed and, therefore, there is no further subgame to analyse. At this point, the delayed promotion to the highest priority $p^\star$ recorded

in $\widetilde{\mathtt{r}}$ is selected and all promotions of the same parity are applied at once. This is done by first moving all regions from $\widetilde{\mathtt{r}}$ into $\mathtt{r}$ and then removing from the resulting function the regions of opposite parity *w.r.t.* $p^\star$, exactly as done by PP+. The search, then, resumes at priority $p^\star$. Intuitively, a promotion is considered as locked if its target priority is either *(a)* greater than some priority in P of opposite parity, which would be otherwise reset, or *(b)* lower than the target of some previously delayed promotion recorded in $\widetilde{\mathtt{r}}$, but greater than the corresponding priority set in $\mathtt{r}$.

The latter condition is required to ensure that the union of a region in $\mathtt{r}$ together with the corresponding recorded region in $\widetilde{\mathtt{r}}$ is still a region. Observe that the whole approach is crucially based on the fact that when a promotion is performed all the regions having lower measure but the same parity are preserved. If this were not the case, we would have no criteria to determine which promotions need to be locked and which, instead, can be freely performed.

	1	2	3	4
6	$\mathtt{a}\downarrow$	...	...	$\mathtt{a},\mathtt{b},\mathtt{d},\mathtt{g},\mathtt{i},\mathtt{j}\downarrow$
5	$\mathtt{b},\mathtt{f},\mathtt{h}\downarrow$	...	...	
4	$\mathtt{c},\mathtt{j}\downarrow$	$\mathtt{c},\mathtt{e},\mathtt{j}\downarrow$	...	$\mathtt{c},\mathtt{e}\uparrow_6$
3	$\mathtt{d}\downarrow$	$\mathtt{d}\downarrow$	$\mathtt{d},\mathtt{g}\not\uparrow_5$	
2	$\mathtt{e}\uparrow_4$			
1		$\mathtt{g}\uparrow_3$		
0			$\mathtt{i}\uparrow_6$	

Table 7.2: DP simulation.

Example 7.3.1. *This idea is summarized by Table 7.2, which contains the execution of the new algorithm on the example in Figure 7.1. The computation proceeds as for* PP+*, until the promotion of the* 1*-region* $\{\mathtt{g}\}$ *shown in Column 2, occurs. This is an instant promotion to* 3*, since the only other promotion already computed and recorded in* P *has value* 4*. Hence, it can be performed and saved in* P *as well. Starting from priority* 3 *in Column 3, the closed* 1*-region* $\{\mathtt{d},\mathtt{g}\}$ *could be promoted to* 5*. However, since its target is greater than* $4 \in \mathrm{P}$*, it is delayed and recorded in* $\widetilde{\mathtt{r}}$*, where it is assigned priority* 5*. At priority* 0*, a delayed promotion of* 0*-region* $\{\mathtt{i}\}$ *to priority* 6 *is encountered and registered, since it would overtake priority* $3 \in \mathrm{P}$*. Now, the resulting subgame is empty. Since the highest delayed promotion is the one to priority* 6 *and no other promotion of the same parity was delayed,* 0*-region* $\{\mathtt{i}\}$ *is promoted and both the auxiliary priority function* $\widetilde{\mathtt{r}}$ *and the set of performed promotions* P *are emptied. The previously computed* 0*-region* $\{\mathtt{c},\mathtt{e},\mathtt{j}\}$ *has the same parity and, therefore, it is not reset, while the positions in both* 1*-regions* $\{\mathtt{b},\mathtt{f},\mathtt{h}\}$ *and* $\{\mathtt{d},\mathtt{g}\}$ *are reset to their original priorities. After maximization of the newly created* 0*-region* $\{\mathtt{a},\mathtt{i}\}$*, positions* $\mathtt{b}$*,* $\mathtt{d}$*,* $\mathtt{g}$*, and* $\mathtt{j}$ *get attracted as well. This leads to the first cell of Column 4, where* 0*-region* $\{\mathtt{a},\mathtt{b},\mathtt{d},\mathtt{g},\mathtt{i},\mathtt{j}\}$ *is open. The next priority to process is* 4*, where* 0*-region* $\{\mathtt{c},\mathtt{e}\}$*, the previous* 0*-region* $\{\mathtt{c},\mathtt{e},\mathtt{j}\}$ *purged of position* $\mathtt{j}$*, is now closed in the corresponding*

subgame and gets promoted to 6. *This results in a* 0*-region closed in the entire game, hence, a dominion for player* 0 *has been found. Note that postponing the promotion of* 1*-region* $\{\mathtt{d},\mathtt{g}\}$ *allowed a reduction in the number of operations. Indeed, the redundant maximization of* 1*-region* $\{\mathtt{b},\mathtt{f},\mathtt{h}\}$ *is avoided.*

It is worth noting that this procedure only requires a linear number of promotions, precisely $\lfloor \frac{h+1}{2} \rfloor$, on the lower bound game $\mathcal{D}_h^{\mathrm{PP}+}$ for PP+. This is due to the fact that all resets are performed on regions that are not destination of any promotion.

7.3.1 DP Dominion Space

As it should be clear from the above informal description, the delayed promotion mechanism is essentially a refinement of the one employed in PP+. Indeed, the two approaches share all the requirements on the corresponding components of a state, on the orderings, and on compatibility relations. However, DP introduces in the state two supplementary elements, a partial priority function $\widetilde{\mathsf{r}}$, which collects the delayed promotions that were not performed on the region function r, and a set of priorities P, which collects the targets of the instant promotions performed. Hence, in order to formally define the corresponding dominion space, we need to provide suitable constraints connecting them with the other components of the search space. The role of function $\widetilde{\mathsf{r}}$ is to record the delayed promotions obtained by moving the corresponding positions from their priority p in r to the new measure q in $\widetilde{\mathsf{r}}$. Therefore, as dictated by Proposition 7.2.2, the union of $\mathsf{r}^{-1}(q)$ and $\widetilde{\mathsf{r}}^{-1}(q)$ must always be a region in the subgame $\mathcal{D}_{\mathsf{r}}^{\leq q}$. In addition, $\widetilde{\mathsf{r}}^{-1}(q)$ can only contain positions whose measure in r is of the same parity as q and recorded in r at some lower priority greater than the current one p. Formally, we say that a partial function $\widetilde{\mathsf{r}} \in \Delta^{\rightharpoonup} \triangleq \mathrm{Ps} \rightharpoonup \mathrm{Pr}$ is *aligned* with a region function r *w.r.t.* p if, for all priorities $q \in \mathsf{rng}(\widetilde{\mathsf{r}})$, it holds that *(a)* $\widetilde{\mathsf{r}}^{-1}(q) \subseteq \mathsf{dom}\big(\mathsf{r}^{(>p)\wedge(<q)\wedge(\equiv_2 q)}\big)$ and *(b)* $\mathsf{r}^{-1}(q) \cup \widetilde{\mathsf{r}}^{-1}(q)$ is an $\wp$-region in $\mathcal{D}_{\mathsf{r}}^{\leq q}$ with $\wp \equiv_2 q$. The state space for DP is, therefore, defined as follows.

Definition 7.3.1 (State Space for DP)**.** *A* DP state space *is a tuple* $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$*, where:*

1. $\mathrm{S} \subseteq \mathrm{S}^{\mathrm{PP}+} \times \Delta^{\rightharpoonup} \times 2^{\mathrm{Pr}}$ *is the set of all triples* $s \triangleq ((\mathsf{r},p),\widetilde{\mathsf{r}},\mathrm{P})$*, called* states*, composed by a* PP+ *state* $(\mathsf{r},p) \in \mathrm{S}^{\mathrm{PP}+}$*, a partial priority function* $\widetilde{\mathsf{r}} \in \Delta^{\rightharpoonup}$ *aligned with* r w.r.t. p*, and a set of priorities* $\mathrm{P} \subseteq \mathrm{Pr}$*.*
2. $\top \triangleq (\top^{\mathrm{PP}+}, \varnothing, \emptyset)$*;*
3. $s_1 \prec s_2$ *iff* $\widehat{s}_1 \prec^{\mathrm{PP}+} \widehat{s}_2$*, for any two states* $s_1 \triangleq (\widehat{s}_1, \_, \_), s_2 \triangleq (\widehat{s}_2, \_, \_) \in \mathrm{S}$*.*

The second property we need to enforce is expressed in the compatibility relation connecting the query and successor functions for DP and regards the closed region pairs that are locked *w.r.t.* the current state. As stated above, a promotion is considered locked if its target priority is either *(a)* greater than some priority in P of opposite parity or *(b)* lower than the target of some previously delayed promotion recorded in $\widetilde{\mathsf{r}}$, but greater than the corresponding priority set in r. Condition *(a)* is the one characterizing the delayed promotion approach, as it reduces the number of resets of previously promoted regions. The two conditions are expressed by the following two formulas, respectively, where q is the target priority of the blocked promotion.

$$\phi_a(q, \mathrm{P}) \triangleq \exists l \in \mathrm{P} \,.\, l \not\equiv_2 q \wedge l < q$$
$$\phi_b(q, \mathsf{r}, \widetilde{\mathsf{r}}) \triangleq \exists v \in \mathsf{dom}(\widetilde{\mathsf{r}}) \,.\, \mathsf{r}(v) < q \leq \widetilde{\mathsf{r}}(v)$$

Hence, an $\wp$-region R is called *$\wp$-locked w.r.t.* a state $s \triangleq ((\mathsf{r}, p), \widetilde{\mathsf{r}}, \mathrm{P})$ if the predicate $\phi_{Lck}(q, s) \triangleq \phi_a(q, \mathrm{P}) \vee \phi_b(q, \mathsf{r}, \widetilde{\mathsf{r}})$ is satisfied, where $q = \mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r} \uplus \widetilde{\mathsf{r}})$ is the best escape priority of the region R for player $\overline{\wp}$ *w.r.t.* the combined region function $\mathsf{r} \uplus \widetilde{\mathsf{r}}$. In addition to the compatibility constraints for PP+, the compatibility relation for DP requires that any $\wp$-locked region, possibly returned by the query function, be maximal and contains the region $\mathsf{r}^{-1}(p)$ associated to the priority p of the current state.

Definition 7.3.2 (Compatibility Relation for DP)**.** *An open quasi dominion pair* $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ *is* compatible *with a state* $s \triangleq ((\mathsf{r}, p), \_, \_) \in \mathrm{S}$*, in symbols* $s \succ (\mathrm{R}, \wp)$*, iff* (1) $(\mathrm{R}, \wp) \in \mathrm{Rg}_{\mathfrak{D}_s}$ *and* (2) *if* R *is* $\wp$*-open in* $\mathfrak{D}_s$ *or it is* $\wp$*-locked* w.r.t. s *then* $\mathrm{R} = \mathsf{atr}^{\wp}_{\mathfrak{D}_s}(\mathsf{r}^{-1}(p))$.

Algorithm 15 implements the successor function for DP. The pseudo-code on the right-hand side consists of three macros used by the algorithm, namely *#Assignment*, *#InstantPromotion*, and *#DelayedPromotion*. The first macro *#Assignment*(ξ) performs the insertion of a new region R into the region function r. In presence of a blocked promotion, *i.e.*, when the parameter ξ is set to `f`, the region is also recorded in $\widetilde{\mathsf{r}}$ at the target priority of the promotion. The macro *#InstantPromotion* corresponds to the DP version of the standard promotion operation of PP+. The only difference is that it must also take care of updating the supplementary elements $\widetilde{\mathsf{r}}$ and P. The macro *#DelayedPromotion*, instead, is responsible for the delayed promotion operation specific to DP.

Algorithm 15: Successor Function.

signature $\downarrow : \succ \to (\Delta \times \mathrm{Pr}) \times \Delta^{\rightharpoonup} \times 2^{\mathrm{Pr}}$

function $s\downarrow(\mathrm{R}, \wp)$

let $((\mathsf{r}, p), \widetilde{\mathsf{r}}, \mathrm{P}) = s$ **in**

1 **if** $(\mathrm{R}, \wp) \in \mathrm{Rg}^{-}_{\ni_s}$ **then**

2-5 $\quad \#Assignment(\mathtt{t})$

else

6 $\quad q \leftarrow \mathsf{bep}^{\overline{\wp}}[\mathrm{R}, \mathsf{r} \uplus \widetilde{\mathsf{r}}]$

7 $\quad$ **if** $\phi_{Lck}(q, s)$ **then**

8 $\quad\quad \widehat{\mathsf{r}} \leftarrow \widetilde{\mathsf{r}}[\mathrm{R} \mapsto q]$

9 $\quad\quad$ **if** $\mathrm{R} \neq \mathrm{Ps}_{\ni_s}$ **then**

10-13 $\quad\quad\quad \#Assignment(\mathtt{f})$

$\quad\quad$ **else**

14-17 $\quad\quad\quad \#DelayedPromotion$

$\quad$ **else**

18-21 $\quad\quad \#InstantPromotion$

22 **return** $((\mathsf{r}^\star, p^\star), \widetilde{\mathsf{r}}^\star, \mathrm{P}^\star)$

Assignment & Promotion Macros.

macro $\#Assignment(\xi)$

1 $\mathsf{r}^\star \leftarrow \mathsf{r}[\mathrm{R} \mapsto p]$

2 $p^\star \leftarrow \mathsf{max}(\mathsf{rng}\left(\mathsf{r}^{\star(<p)}\right))$

3 $\widetilde{\mathsf{r}}^\star \leftarrow$ **#if** ξ **#then** $\widetilde{\mathsf{r}}$ **#else** $\widehat{\mathsf{r}}$

4 $\mathrm{P}^\star \leftarrow \mathrm{P}$

macro $\#DelayedPromotion$

1 $p^\star \leftarrow \mathsf{max}(\mathsf{rng}(\widehat{\mathsf{r}}))$

2 $\mathsf{r}^\star \leftarrow \mathsf{pr} \uplus (\mathsf{r} \uplus \widehat{\mathsf{r}})^{(\geq p^\star) \vee (\equiv_2 p^\star)}$

3 $\widetilde{\mathsf{r}}^\star \leftarrow \emptyset$

4 $\mathrm{P}^\star \leftarrow \emptyset$

macro $\#InstantPromotion$

1 $p^\star \leftarrow q$

2 $\mathsf{r}^\star \leftarrow \mathsf{pr} \uplus \mathsf{r}^{(\geq p^\star) \vee (\equiv_2 p^\star)}[\mathrm{R} \mapsto p^\star]$

3 $\widetilde{\mathsf{r}}^\star \leftarrow \widetilde{\mathsf{r}}^{(> p^\star)}$

4 $\mathrm{P}^\star \leftarrow \mathrm{P} \cap \mathsf{rng}(\mathsf{r}^\star) \cup \{p^\star\}$

If the current region R is open in the subgame $\ni_s$, the main algorithm proceeds, similarly to Algorithm 14, at assigning to it the current priority p in r. This is done by calling macro $\#Assignment$ with parameter $\mathtt{t}$. Otherwise, the region is closed and a promotion should be performed at priority q, corresponding to the bep of that region *w.r.t.* the composed region function $\mathsf{r} \uplus \widetilde{\mathsf{r}}$. In this case, the algorithm first checks whether such promotion is locked *w.r.t.* s at Line 7. If this is not the case, then the promotion is performed as in PP+, by executing $\#InstantPromotion$, and the target is kept track of in the set P. If, instead, the promotion to q is locked, but some portion of the game still has to be processed, the region is assigned its measure p in r and the promotion to q is delayed and stored in $\widetilde{\mathsf{r}}$. This is done by executing $\#Assignment$ with parameter $\mathtt{f}$. Finally, in case the entire game has been processed, the delayed promotion to the highest priority recorded in $\widetilde{\mathsf{r}}$ is selected and applied. The macro $\#DelayedPromotion$ is executed, thus merging r with $\widetilde{\mathsf{r}}$. Function $\widetilde{\mathsf{r}}$ and set P are, then, erased, in order to begin a new round of the search. Observe that, when a promotion is performed, whether instant or delayed, we always preserve the underlying regions of the same parity, as done by the PP+

algorithm. This is a crucial step in order to avoid the pathological exponential worst case for the original PP procedure.

The soundness of the solution procedure relies on the following theorem.

Theorem 7.3.1 (DP Dominion Space)**.** *For a game* ∂*, the* DP structure $\mathcal{D} \triangleq \langle \partial, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where* $\mathcal{S}$ *is given in Definition 7.3.1,* $\succ$ *is the relation of Definition 7.3.2, and* $\Re$ *and* $\downarrow$ *are the functions computed by Algorithms 13 and 15, where in the former the assumption "***let** $(\mathsf{r}, p) = s$*" is replaced by "***let** $((\mathsf{r}, p), \_, \_) = s$*" is a dominion space.*

To prove the above theorem, we have to show that the three components $\mathcal{S}$, $\Re$, and $\downarrow$ of the structure $\mathcal{D}$ satisfy the properties required by Definition 8.2.2 of dominion space. We do this through the Lemmas 7.3.1, 7.3.2, and 7.3.3.

Lemma 7.3.1 (State Space)**.** *The* DP *state space* $\mathcal{S} = \langle \mathrm{S}, \top, \prec \rangle$ *for a game* $\partial \in \mathcal{P}$ *is a well-founded partial order* w.r.t. $\prec$ *with designated element* $\top \in \mathrm{S}$*.*

Proof. Due to Definition 7.3.1 and Lemma 7.2.1, to prove that $\mathcal{S}$ satisfies the required properties, we have only to observe that the distinguished element $\top \triangleq (\top^{\mathrm{PP}+}, \varnothing, \emptyset)$ is a state. This immediately follows from the fact that $\top^{\mathrm{PP}+} = (\mathsf{pr}, \mathsf{pr}(\partial))$ is a PP+ state and the empty supplementary function $\varnothing$ is trivially aligned with pr *w.r.t.* $\mathsf{pr}(\partial)$. □

Lemma 7.3.2 (Query Function)**.** *The function* $\Re$ *is a query function,* i.e.*, for all states* $s \in \mathrm{S}$*, it holds that* (1) $\Re(s) \in \mathrm{QD}$ *and* (2) *if* $\Re(s) \in \mathrm{QD}^{-}$ *then* $s \succ \Re(s)$*.*

Proof. Due to Definition 7.3.2 and Lemma 7.2.2, the thesis directly follows once observed that $\mathrm{R} = \mathsf{atr}^{\wp}_{\partial_s}[\mathsf{r}^{-1}(p)]$ independently from the fact that it is $\wp$-open in ∂_s or $\wp$-locked *w.r.t.* s. □

Lemma 7.3.3 (Successor Function)**.** *The function* $\downarrow$ *is a successor function,* i.e.*, for all states* $s \in \mathrm{S}$ *and quasi dominion pairs* $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ *with* $s \succ (\mathrm{R}, \wp)$*, it holds that* (1) $s{\downarrow}(\mathrm{R}, \wp) \in \mathrm{S}$ *and* (2) $s{\downarrow}(\mathrm{R}, \wp) \prec s$*.*

Proof. Let $s \triangleq ((\mathsf{r}, p), \widetilde{\mathsf{r}}, \mathrm{P}) \in \mathrm{S}$ be a state, $(\mathrm{R}, \wp) \in \mathrm{QD}^{-}$ an open quasi dominion pair in ∂ compatible with s, and $s^{\star} = ((\mathsf{r}^{\star}, p^{\star}), \widetilde{\mathsf{r}}^{\star}, \mathrm{P}^{\star}) \triangleq s{\downarrow}(\mathrm{R}, \wp)$ the result obtained by computing the function $\downarrow$ on s and $(\mathrm{R}, \wp)$.

It is quite easy to see that $(\mathsf{r}^{\star}, p^{\star}) \in \mathrm{S}^{\mathrm{PP}+}$ and $(\mathsf{r}^{\star}, p^{\star}) \prec^{\mathrm{PP}+} (\mathsf{r}, p)$. Indeed, in case one of the two macros $\#Assignment(\xi)$ with $\xi \in \{\mathtt{f}, \mathtt{t}\}$ and $\#InstantPromotion$ of Algorithm 15 is executed, one can derive the thesis by applying exactly the same reasoning used in the proof of Lemma 7.2.3, since the instructions concerning the two components $\mathsf{r}^{\star}$ and $p^{\star}$ are those used in Algorithm 14. If the

#DelayedPromotion macro is considered, instead, due to Item 1 of Definition 7.3.1, $\widetilde{\mathsf{r}}$ is aligned with r *w.r.t.* p, so, the set of positions $\mathsf{r}^{-1}(q) \cup \widetilde{\mathsf{r}}^{-1}(q)$ is a β-region in $\mathcal{D}_{\mathsf{r}}^{\leq q}$, for all $q \in \mathsf{rng}(\widetilde{\mathsf{r}})$ with $\beta \equiv_2 q \equiv_2 p^\star$. Hence, by applying the same reasoning used for the *#InstantPromotion* macro, the thesis follows in this case as well.

To conclude the proof, we need to show that $\widetilde{\mathsf{r}}^\star$ is aligned with $\mathsf{r}^\star$ *w.r.t.* $p^\star$. Again, we do this by means of a case analysis on the four possible macros.

- *#Assignment*($\mathtt{f}$). Lines 2 and 3 of the macro ensure that $p^\star < p$ and $\widetilde{\mathsf{r}}^\star = \widetilde{\mathsf{r}}$. Hence, the property follows from the fact that $\widetilde{\mathsf{r}}$ is aligned with r *w.r.t.* p.

- *#Assignment*($\mathtt{t}$). By Lines 2 and 3, $p^\star < p$ and $\widetilde{\mathsf{r}}^\star = \widetilde{\mathsf{r}}[\mathrm{R} \mapsto q]$. Since $\widetilde{\mathsf{r}}$ is aligned with r *w.r.t.* p, we have that $\widetilde{\mathsf{r}}^{-1}(z) \subseteq \mathsf{dom}\big(\mathsf{r}^{(>p)\wedge(<z)\wedge(\equiv_2 z)}\big)$, and $\mathsf{r}^{-1}(z) \cup \widetilde{\mathsf{r}}^{-1}(z)$ is an $\wp$-region in $\mathcal{D}_{\mathsf{r}}^{\leq z}$, for all priorities $z \in \mathsf{rng}(\widetilde{\mathsf{r}})$. Now, if $z \neq q \triangleq \mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r} \uplus \widetilde{\mathsf{r}})$, we have $\widetilde{\mathsf{r}}^{\star\,-1}(z) = \widetilde{\mathsf{r}}^{-1}(z)$. So, the required properties on $\widetilde{\mathsf{r}}^{\star\,-1}(z)$ are immediately satisfied. If $z = q$, instead, the $\wp$-region R is added to both r and $\widetilde{\mathsf{r}}$ with measures p and z, respectively. Hence, $\widetilde{\mathsf{r}}^{\star\,-1}(z) \subseteq \mathsf{dom}\big(\mathsf{r}^{\star(>p^\star)\wedge(<z)\wedge(\equiv_2 z)}\big)$ is verified, since $p \equiv_2 z$. Finally, we need to show that $\mathsf{r}^{\star\,-1}(z) \cup \widetilde{\mathsf{r}}^{\star\,-1}(z)$ is an $\wp$-region in $\mathcal{D}_{\mathsf{r}^\star}^{\leq z}$. This follows from Proposition 7.2.2 applied to $\mathsf{r}^{-1}(z) \cup \widetilde{\mathsf{r}}^{-1}(z)$ and R, since R is an $\wp$-dominion in the subgame $\mathcal{D}_{\mathsf{r}^\star}^{\leq z} \setminus (\mathsf{r}^{-1}(z) \cup \widetilde{\mathsf{r}}^{-1}(z))$, due to the fact that $\widetilde{\mathsf{r}}^{-1}(z) \subseteq \mathsf{dom}\big(\mathsf{r}^{(>p)\wedge(<z)\wedge(\equiv_2 z)}\big)$.

- *#DelayedPromotion*. The property is trivially satisfied, since $\widetilde{\mathsf{r}}^\star = \varnothing$.

- *#InstantPromotion*. Since $\widetilde{\mathsf{r}}^{\star(\leq p^\star)} = \varnothing$ and $\widetilde{\mathsf{r}}^{\star(>p^\star)} = \widetilde{\mathsf{r}}^{(>p^\star)}$, the property follows from the fact that $\widetilde{\mathsf{r}}$ is aligned with r *w.r.t.* p.

□

It is immediate to observe that the following mapping $\mathsf{h} : ((\mathsf{r}, p), \_, \_) \in \mathrm{S}^{\mathrm{DP}} \mapsto (\mathsf{r}, p) \in \mathrm{S}^{\mathrm{PP}+}$, which takes DP states to PP+ states by simply forgetting the additional elements $\widetilde{\mathsf{r}}$ and P, is a homomorphism. This, together with a trivial calculation of the number of possible states, leads to the following theorem.

Theorem 7.3.2 (DP Size & Depth Upper Bounds)**.** *The size of the* DP *dominion space* $\mathcal{D}_{\mathcal{D}}$ *for a game* $\mathcal{D} \in \mathcal{P}$ *with* $n \in \mathbb{N}_+$ *positions and* $k \in [1, n]$ *priorities is bounded by* $2^k k^{2n}$*. Moreover, its depth is not greater then the one of the* PP+ *dominion space* $\mathcal{D}_{\mathcal{D}}^{\mathrm{PP}+}$ *for the same game.*

Unfortunately, also this improved promotion policy happens to suffer from exponential behaviors. However, the exponential lower-bound family we found has a much more complex structure than the one previously described for PP+. In particular, it requires multiple positions with the same priority in

order to fool the delayed promotion criterion and force the algorithm to behave exactly as PP+. There seems to be no obvious way to enforce a similar behavior on games having a single position for each priority. Since it is well known that any parity game can be transformed into an equivalent one with as many priorities as positions [Sti01], the DP approach can still be considered as a candidate to a polynomial-time solution of the problem.

In this family, the game $\mathfrak{D}_h^{\mathrm{DP}}$ of index $h \geq 1$, where h denotes the number of chains it contains, is defined as follows. For all indexes $i \in [1, h]$ and $j \in [i-1, h-(i \bmod 2)] \cup \{h+i-1+(h \bmod 2)\}$, there is a position (i, j), which we shortly denote by j_i. Position j_i is connected to the i-th chain, belongs to player $i \bmod 2$, *i.e.*, $j_i \in \mathrm{Ps}^{(i \bmod 2)}$, and has priority j, *i.e.*, $\mathsf{pr}(j_i) = j$. For any chain $i \in]1, h]$, the head position $(h+i-1+(h \bmod 2))_i$ has a unique move to the head $(h+i-2+(h \bmod 2))_{i-1}$ of the lower chain $i-1$. Moreover, the head $(h+(h \bmod 2))_1$ of the first chain is connected to the tails $(h-1)_i$ of all the chains of odd index, *i.e.*, with $i \equiv_2 1$. The lower position $(i-1)_i$ in each chain has a self-move plus one move to the head $(h+i-1+(h \bmod 2))_i$. Finally, each position j_i, with $j \in [i, h-(i \bmod 2)]$, has a unique move to a lower position $(j-1)_i$ in the same chain. Figure 7.3 depicts the instance of $\mathfrak{D}_h^{\mathrm{DP}}$ with $h = 4$.

Similarly to the execution depth of the PP+ dominion space of the game in Figure 7.2, the execution depth of the DP dominion space for this game is exponential in h. Indeed, the promotion operations performed on each chain, merging its second position to its head, can simulate the increments of a partial binary counter with h digits. The number of configurations of this counter for the DP algorithm can be proved to be $\mathsf{P}(h) + \lceil n/2 \rceil - 2$, where P is the function counting the number of configurations for the PP+ shown in the proof of Theorem 7.2.2.

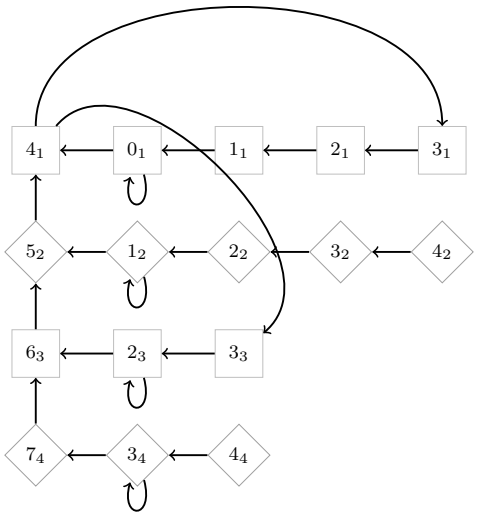


Figure 7.3: The $\mathfrak{D}_4^{\mathrm{DP}}$ game.

The search procedure on $\mathfrak{D}_4^{\mathrm{DP}}$ starts by building the following seven open regions: the 1-region $\{7_4\}$, the 0-region $\{6_3\}$, the 1-region $\{5_2\}$, the 0-region $\{4_1, 4_2, 4_4\}$, the 1-region $\{3_1, 3_2, 3_3, 3_4\}$, the 0-region $\{2_1, 2_2, 2_3\}$, and the 1-region $\{1_1, 1_2, \}$. This state represents the configuration of the counter, where all four digits are set to 0. The closed 0-region $\{0_1\}$ is then found and promoted to 4. Now, the counter is set to 0001. After that, the 1-region $\{3_2, 3_3, 3_4\}$ is built again followed by the 0-region $\{2_2, 2_3\}$. Next, the

closed 1-region $\{1_2\}$ is computed, and is promoted to 5. Due to the reset criterion, the positions in the 0-region with priority 4 are reset to their original priority, as they belong to the opponent player *w.r.t.* to the promoted region. This releases the chain with head 4, which corresponds to the reset of the least significant digit of the counter caused by the increment of the second one, *i.e.*, the counter displays 0010. The search resumes at priority 5 and the 0-region $\{4_1, 4_4\}$, the 1-region $\{3_1, 3_3, 3_4\}$, the 0-region $\{2_1, 2_3\}$, the 1-region $\{1_1\}$, and the 0-region $\{0_1\}$ are computed once again. A second promotion of $\{0_1\}$ to 4 is performed, resulting in the counter assuming value 0011. When the closed 0-region $\{2_3\}$ is promoted to 6, however, only the 1-region $\{5_2, 1_2, 2_2, 3_2, 4_2\}$ is reset, leading to configuration 0101. Hence, configuration 0100 is skipped. Similarly, when, the counter reaches configuration 0111 and 1-region $\{3_4\}$ is promoted to 7, the 0-regions $\{4_1, 0_1, 1_1, 2_1, 3_1\}$ and $\{6_3, 2_3, 3_3\}$ are reset, leaving 1-region $\{5_2, 1_2, 2_2, 3_2, 4_2\}$ intact. This leads directly to configuration 1010 of the counter, skipping configurations 1000 and 1001.

Observe that for each configuration, but the initial one, a promotion is required. So the number of promotions to reach the all-1 configuration is $\mathsf{Fib}(h+3)-1$. Once the last configuration is reached, the region of the least significant digit is promoted to the region corresponding to the third significant digit, which is, in turn, promoted to the fifth significant digit and so on, until a region containing all positions of the game, but those of the most significant digit, if h is even, is obtained. Such region is a dominion and the algorithm terminates. The number of these additional promotions is exactly $\lceil n/2 \rceil - 2$. Hence, the total number is $\mathsf{Fib}(h+3) - 1 + \lceil n/2 \rceil - 2$. The observation above allows us to provide an estimation of the depth of the DP dominion space.

Theorem 7.3.3 (Execution-Depth Lower Bound)**.** *For all $h \in \mathbb{N}$, there exists a* DP *dominion space $\mathcal{D}_h^{\mathrm{DP}}$ with $n = 2h + (h^2 - h \bmod 2)/2$ positions and $k = 2h$ priorities, whose execution depth is at least $\mathsf{Fib}(h+3) - 3 + \lceil n/2 \rceil = \Theta\Big(((1+\sqrt{5})/2)^{\sqrt{n}}\Big)$.*

Note that in a game $\mathcal{D}_h^{\mathrm{DP}}$, the number of positions is quadratic in the number of priorities, hence, $k = \mathrm{O}(\sqrt{n})$. This appears to be a crucial element in order to force the DP algorithm to behave as PP+. Indeed, it is essential for every region of priority $p \in [1, k-1]$ to be open until all the lower ones with priorities $p' < p$ are promoted to the corresponding measure $h + p' + (h \bmod 2)$. The only way to do this seems to require $p+1$ positions with the same parity p.

7.4 Experimental Evaluation

The technique proposed in the paper has been implemented in PGSolver [FL09], a tool that collects implementations of several parity game solvers proposed in the literature. The tool provides benchmarking

Benchmark	Positions	*DomDec*	*BigStp*	*StrImp*	*Rec*	PP	PP+	DP
Hanoi	$6.3 \cdot 10^6$	21.4	21.4	‡	17.4	**7.0**	**7.0**	7.4
Elevator	$7.7 \cdot 10^6$	†	‡	‡	‡	19.8	**19.7**	20.4
Lang. Incl.	$5 \cdot 10^6$	†	‡	‡	145.5	**16.5**	**16.5**	**16.5**
Ladder	$4 \cdot 10^6$	†	‡	‡	35.0	**7.9**	**7.9**	8.1
Str. Imp.	$4.5 \cdot 10^6$	81.0	82.8	†	71.0	**57.0**	57.1	57.3
Clique	$8 \cdot 10^3$	†	‡	†	†	**10.8**	10.9	**10.8**
MC. Lad.	$7.5 \cdot 10^6$	†	‡	‡	**4.3**	4.4	**4.3**	4.5
Rec. Lad.	$5 \cdot 10^4$	†	‡	‡	‡	**62.8**	63.0	64.9
Jurdziński	$4 \cdot 10^4$	†	†	188.2	†	**69.6**	69.7	71.8
WC. Rec.	$3 \cdot 10^4$	†	†	**9.4**	†	10.2	10.2	18.2

Table 7.3: Execution times in seconds on several benchmark families.
Time out (†) is set to 600 seconds and memory out (‡) to 7.5Gb.

tools that can be used to evaluate the performance of the solvers on both concrete and synthetic benchmarks. The concrete ones include three classes of games, namely Towers-of-Hanoi, Elevator-Verification and Language-Inclusion, which encode standard verification problems. The synthetic ones, instead, provide both worst-case exponential families for the solvers included in the tool and random games generators. The solvers considered in the experiments include the original PP algorithm [1], the two versions PP+ and DP presented in this paper, the Recursive algorithm *Rec* [Zie98], the two Dominion Decomposition algorithms *DomDec* [JPZ08] and *BigStp* [Sch07], and the Strategy Improvement algorithm *StrImp* [VJ00].

We also experimented with Small Progress Measure [Jur00] and with the two quasi-polynomial solvers recently proposed in [FJS+17] and [JL17]. These last three solvers, however, were unable to solve any of the benchmarks considered within the time limit and have been left out from the following experimental evaluation.

Table 7.3 reports the results of the solvers on the benchmark families available in PGSolver. We only report on the biggest instances we could deal with, given the available computational resources [2]. The parameter Positions gives the number of positions in the games and the best performance are emphasized in bold.[3] The first three rows consider the concrete verification problems mentioned above.

[1] The version of PP used in the experiments is actually an improved implementation of the one described in [BDM16c].

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

[3] The instances were generated by issuing the following PGSolver commands:

`towersofhanoi 13`, `elevatorgame 8`, `langincl 500 100`, `cliquegame 8000`, `laddergame 4000000`,

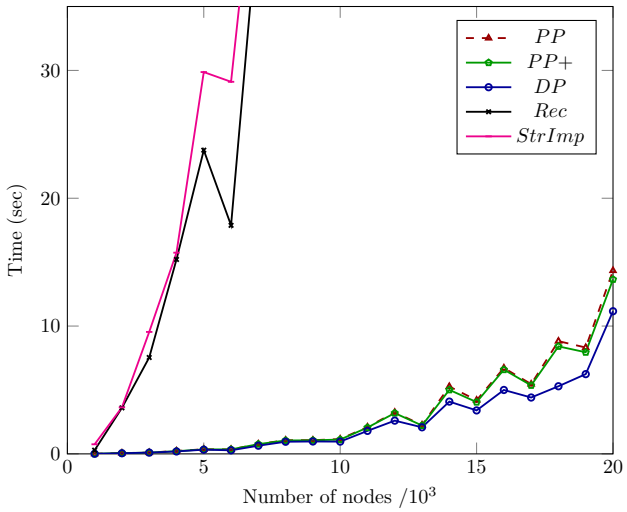


Figure 7.4: Solution times on random games from [BDM18b].

On the Tower-of-Hanoi problem all the solvers perform reasonably well, except for *StrImp* due its high memory requirements. The Elevator-Verification problem proved to be very demanding in terms of memory for all the solvers, except for the priority-promotion based algorithms and *DomDec*, which, however, could not solve it within the time limit of 10 minutes.

Our solvers perform extremely well on both this benchmark and on Language Inclusion, which could be solved only by *Rec* among the other solvers.

On the worst case benchmarks, they all perform quite well on Ladder, Strategy Improvement, Jurdziński, Recursive Ladder, and even on Clique, which proved to be considerably difficult for all the other solvers. The Modelchecker game was essentially a tie with *Rec*. The only family on which they were slightly outperformed by the Strategy Improvement algorithm is the new worst case family *WC-Rec*, which was introduced as a robust worst-case for various solvers in [BDM17]. On these benchmarks the new algorithms exhibit the most consistent behavior overall. Indeed, on all those families the priority-promotion based algorithms perform no promotions, regardless of the input parameters, except for the Elevator-Verification problem, where they require only two promotions. This constant bound on the number of promotions implies that, for those games, the three solvers, PP, PP+, and DP only require time linear in the size of the game and the number of priorities, which explains the similarity of their performance. It is worth noting that, when few promotions are involved, DP is likely to suffer from some overhead *w.r.t.* both PP and PP+. When, on the other hand, the number of promotions increases,

`stratimprgen -pg friedmannsubexp 1000`, `modelcheckerladder 2500000`, `recursiveladder 10000`, and `jurdzinskigame 100 100`.

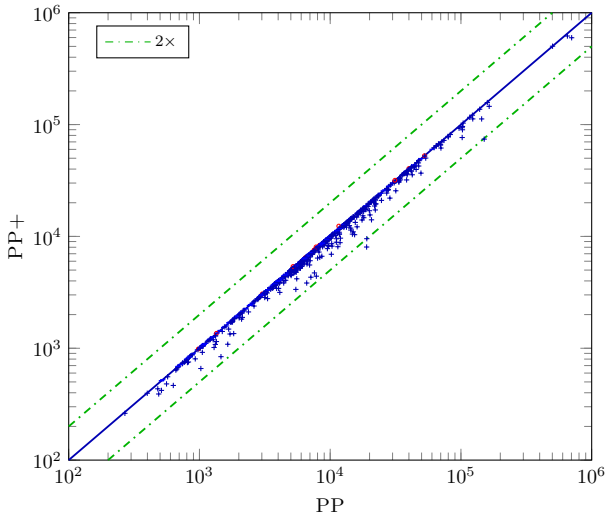


Figure 7.5: Number of promotions: comparison between PP and PP+ on random games with 50000 positions.

the overhead is often compensated by the reduction of the number of promotions required to solve the game, as witnessed by the experiments on random games described below.

Figure 7.4 compares the running times of the new algorithms, PP+ and DP, against the original PP and the solvers *Rec* and *StrImp* on randomly generated games. The other two solvers *DomDec* and *BigStep* perform quite poorly on those games, hitting the time-out already for very small instances. This first pool of benchmarks contains the same games used in the experimental evaluation presented in [BDM18b]. It contains 2000 random games of size ranging from 1000 to 20000 positions and 2 outgoing moves per position. Interestingly, random games with very few moves prove to be much more challenging for the priority promotion based approaches than those with a higher number of moves per position, and often require a much higher number of promotions.

Since the behavior of the solvers is typically highly variable, even on games of the same size and priorities, to summaries the results we took the average running time on clusters of games. Therefore, each point in the graph shows the average time over a cluster of 100 different games of the same size: for each size value n, we chose the numbers $k = n \cdot i/10$ of priorities, with $i \in [1, 10]$, and 10 random games were generated for each pair n and k. We set a time-out to 180 seconds (3 minutes). Solver PP+ performs slightly better than PP, while DP shows a much more convincing improvement on the average time.

Similar experiments were also conducted on random games with a higher number of moves per

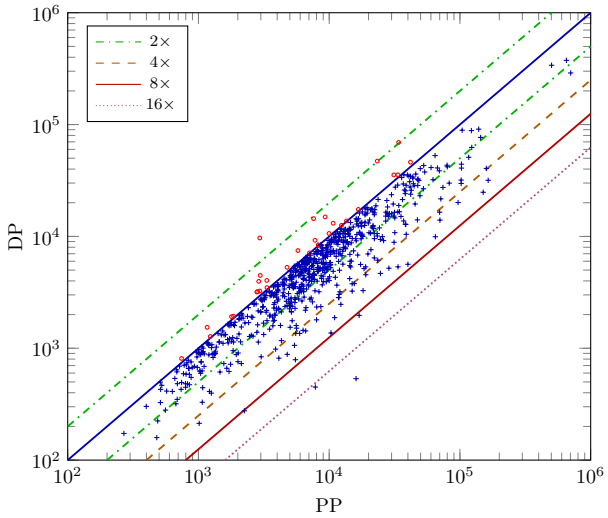


Figure 7.6: Number of promotions: comparison between PP and DP on random games with 50000 positions.

position and up to 1000000 positions. The resulting games turn out to be very easy to solve by all the priority promotion based approaches, requiring few seconds only. The reason seems to be that the higher number of moves significantly increases the dimension of the computed regions and, consequently, also the chances to find a closed one. Indeed, the number of promotions required by PP+ and DP on all those games is typically zero, and the whole solution time is due exclusively to a very limited number of attractors needed to compute the few regions contained in the games. The only other solver that can easily solve these games is *Rec*, whose performance is only slightly worse than that of the priority-promotion-based approaches.

Since the exponential behaviors of the priority-promotion-based algorithms are tightly connected with the number of promotions needed to solve a game and the main aim is to reduce such a number, we devised specific benchmarks towards analyzing the behavior of the algorithms *w.r.t.* this measure. The second pool of benchmarks contains 740 games, each with 50000 positions, 2 moves per positions and priorities varying from 8000 to 12000. These games are much harder than the previous ones and have been selected among random games, whose solution requires PP more than 500 and up to 700000 promotions to be solved. On these games we measured both the number of promotions performed and the solution times.

Figure 7.5 reports the experimental results comparing the number of promotions performed by PP and PP+ on these games and provides experimental support to the expectation that PP+ almost always

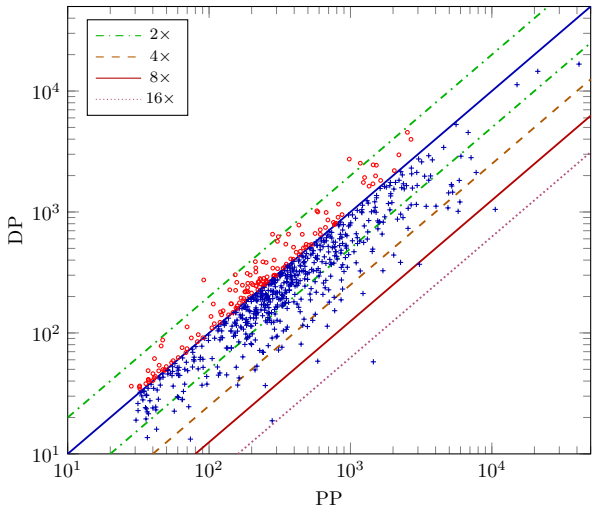


Figure 7.7: Solution time: comparison between PP and DP on random games with 50000 positions.

requires less effort than PP, even though the benefits appear quite limited. This essentially confirms the results reported in Figure 7.4.

On the other hand, Figure 7.6, comparing DP and PP, reveals that DP does reduce the number promotions considerably. The benefits of the new promotion policy appear to be substantial as, in many cases, PP requires between two to eight times as many promotions as DP to solve a game. Notice that the scale in the figures is logarithmic and the diagonal lines are labeled with the corresponding multiplication factor. The figure also shows that, in very few cases, PP does require less promotions than DP. This is due to the fact that the two algorithms typically follow different solution paths within the dominion space and that delaying promotions may defer the discovery of a closed dominion. Nonetheless, the DP policy does pay off significantly on the vast majority of the benchmarks. Finally, Figure 7.7 shows how the reduction on the number of promotions translates into solution times. The results, once again reported on a logarithmic scale, confirm that, despite the additional overhead in the computation of DP that sometimes outweighs the gain in terms of promotions shown in Figure 7.6, DP outperforms PP on most of the benchmarks. Recently, the new tool Oink [vD18] has been presented. The tool, written in C++, provides efficient implementations of several parity games algorithms. Experiments performed with Oink on the same benchmarks used here essentially confirm the results obtained with PGSolver reported above.

Chapter 8

Region Recovery Technique for Parity Games

In this chapter we describe the second refinement of the Priority Promotion approach, which tries to recover some of the information the original approach loses during the computation, speeding up significantly the solution process.

This work has been presented at the

- 12th Haifa Verification Conference (HVC'16), held in November 14-17, 2016, in Haifa, Israel, with the title of "Improving Priority Promotion for Parity Games" and published on Volume 10028 of LNCS, pages 1-17. Springer, 2016.

The content of the chapter is based on [BDM16b].

8.1 Abstract

We propose a new algorithm, called RR for *region recovery*, which is built on top of of PP and is based on a form of conservation property of quasi dominions. This property provides sufficient conditions for a subset a quasi α-dominion to be still a quasi α-dominion. By exploiting this property, the RR algorithm can significantly reduce the execution of the resetting phase, which is now limited to the cases when the conservation property is not guaranteed to hold. For the resulting procedure no exponential worst case has been found yet. Experiments on randomly generated games also show that the new approach performs significantly better than PP in practice, while still preserving the same space complexity.

8.2 Quasi Dominion Approach

This section recalls the notion of *quasi dominion* already introduced in the previous chapters, that an expert reader can simply skip. The priority promotion algorithm proposed in [BDM16c] attacks the problem of solving a parity game $\mathcal{D}$ by computing one of its dominions D, for some player $\wp \in \{0,1\}$, at a time. Indeed, once the $\wp$-attractor $\mathrm{D}^\star$ of D is removed from $\mathcal{D}$, the smaller game $\mathcal{D} \setminus \mathrm{D}^\star$ is obtained, whose positions are winning for one player iff they are winning for the same player in the original game. This allows for decomposing the problem of solving a parity game to that of iteratively finding its dominions [JPZ08].

In order to solve the dominion problem, the idea described in [BDM16c] is to introduce a much weaker notion than that of dominion, called *quasi dominion*, which satisfies, under suitable conditions, a composition property that eventually brings to the construction of a dominion. Intuitively, a quasi $\wp$-dominion Q is a set of positions on which player $\wp$ has a *witness strategy* $\sigma_\wp$, whose induced plays either remain inside Q forever and are winning for $\wp$ or can exit from Q passing through a specific set of escape positions.

Definition 8.2.1 (Quasi Dominion [BDM16c])**.** *Let $\mathcal{D} \in \mathcal{P}$ be a game and $\wp \in \{0,1\}$ a player. A non-empty set of positions $\mathrm{Q} \subseteq \mathrm{Ps}$ is a* quasi $\wp$-dominion *in $\mathcal{D}$ if there exists an $\wp$-strategy $\sigma_\wp \in \mathrm{Str}^\wp(\mathrm{Q})$, called* $\wp$-witness *for Q, such that, for all $\overline{\wp}$-strategies $\sigma_{\overline{\wp}} \in \mathrm{Str}^{\overline{\wp}}(\mathrm{Q})$, with $\mathsf{int}^{\overline{\wp}}(\mathrm{Q}) \subseteq \mathsf{dom}(\sigma_{\overline{\wp}})$, and positions $v \in \mathrm{Q}$, the induced play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ satisfies $\mathsf{pr}(\pi) \equiv_2 \wp$, if π is infinite, and $\mathsf{lst}(\pi) \in \mathsf{esc}^{\overline{\wp}}(\mathrm{Q})$, otherwise.*

Observe that, if all the plays induced by the witness $\sigma_\wp$ remain in the set Q forever, this is actually an $\wp$-dominion and, therefore, a subset of the winning region $\mathrm{Wn}_\wp$ of $\wp$, with $\sigma_\wp$ the projection over Q of some $\wp$-winning strategy on the entire game. In this case, the escape set of Q is empty, *i.e.*, $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q}) = \emptyset$, and Q is said to be $\wp$-*closed*. In general, however, a quasi $\wp$-dominion Q that is not an $\wp$-dominion, *i.e.*, such that $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q}) \neq \emptyset$, need not be a subset of $\mathrm{Wn}_\wp$ and it is called $\wp$-*open*. Indeed, in this case, some induced play may not satisfy the winning condition for that player once exited from Q, *e.g.*, by visiting a cycle containing a position with maximal priority of parity $\overline{\wp}$. The set of triples $(\mathrm{Q}, \sigma, \wp) \in 2^{\mathrm{Ps}} \times \mathrm{Str} \times \{0,1\}$, where Q is a quasi $\wp$-dominion having σ as one of its $\wp$-witnesses, is denoted by QD, and is partitioned into the sets QD^- and QD^+ of open and closed quasi $\wp$-dominion triples, respectively.

Similarly to the other *divide et impera* techniques proposed in the literature, the one reported in [BDM16c], called PP, does not make any algorithmic use of the witness strategy $\sigma_\wp$ associated with a quasi dominion Q, as this notion is only employed in the correctness proof. In this work, instead,

we strongly exploit the effective computability of such a witness in order to considerably alleviate the collateral effects of a *reset operation* required by PP to ensure the soundness of the approach, which is also responsible for the exponential worst cases. Indeed, this algorithm needs to forget previously computed partial results after each compositions of two quasi-dominions, since the information computed during the entire process cannot ensure that these results can be still correctly used in the search for a dominion. In this work, instead, we exploit the following simple observation on the witness strategies, formally reported in Lemma 8.2.1, to determine which partial results can be safely preserved.

In general, quasi $\wp$-dominions are not closed under restriction. For example, consider the quasi 1-dominion $\mathrm{Q}^\star = \{\mathtt{a}, \mathtt{c}, \mathtt{d}\}$ of Figure 8.1 with unique 1-witness strategy $\sigma^\star = \{\mathtt{c} \mapsto \mathtt{a}, \mathtt{d} \mapsto \mathtt{c}\}$ and its subset $\mathrm{Q} = \{\mathtt{d}\}$. It is quite immediate to see that Q is not a quasi 1-dominion, since $\mathsf{esc}^0(\mathrm{Q}) = \emptyset$, but player 1 does not have a strategy that induces infinite 1-winning plays that remain in Q. Indeed, $\sigma^\star$ requires player 1 to exit from Q by going from $\mathtt{d}$ to $\mathtt{c}$. On the contrary, the subset $\mathrm{Q} = \{\mathtt{c}, \mathtt{d}\}$ is still a 1-dominion with 1-witness strategy $\sigma = \{\mathtt{d} \mapsto \mathtt{c}\} = \sigma^\star{\restriction}\mathsf{stay}^1(\mathrm{Q})$, since $\mathsf{esc}^0(\mathrm{Q}) = \{\mathtt{c}\}$. Hence, the restriction σ of $\sigma^\star$ to $\mathsf{stay}^1(\mathrm{Q}) = \{\mathtt{d}\}$ is still a well-defined strategy on Q. Inspired by this observation, we provide the following sufficient criterion for a subset Q of a quasi $\wp$-dominion $\mathrm{Q}^\star$ to be still a quasi $\wp$-dominion.

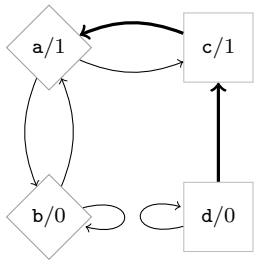


Figure 8.1: Witness strategy.

Lemma 8.2.1 (Witness Strategy)**.** *Given a quasi $\wp$-dominion $\mathrm{Q}^\star$ having $\sigma^\star \in \mathrm{Str}^\wp(\mathrm{Q}^\star)$ as* $\wp$-witness *and a subset of positions $\mathrm{Q} \subseteq \mathrm{Q}^\star$, the restriction $\sigma \triangleq \sigma^\star{\restriction}\mathsf{stay}^\wp(\mathrm{Q})$ is an* $\wp$-witness *for* Q *iff $\sigma \in \mathrm{Str}^\wp(\mathrm{Q})$.*

The proof-idea behind this lemma is very simple. Any infinite play induced by the restriction of σ on Q is necessarily winning for player $\wp$, since it is coherent with the original $\wp$-witness $\sigma^\star$ of $\mathrm{Q}^\star$ as well. Now, if $\sigma \in \mathrm{Str}^\wp(\mathrm{Q})$, we are also sure that any finite play ends in $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q})$, as required by the definition of quasi dominion. Therefore, σ is an $\wp$-witness for Q, which is, then, a quasi $\wp$-dominion. On the other hand, if $\sigma \notin \mathrm{Str}^\wp(\mathrm{Q})$, there exists a finite play induced by σ that does not terminate in $\mathsf{esc}^{\overline{\wp}}(\mathrm{Q})$. Hence, σ is not an $\wp$-witness. In this case, we cannot ensure that Q is a quasi $\wp$-dominion.

The priority promotion algorithm explores a partial order, whose elements, called *states*, record information about the open quasi dominions computed along the way. The initial state of the search is the top element of the order, where the quasi dominions are initialised to the sets of positions with the same priority. At each step, a new quasi $\wp$-dominion Q together with one of its possible $\wp$-witnesses σ is extracted from the current state, by means of a *query* operator $\Re$, and used to compute a successor state, by means of a *successor* operator $\downarrow$, if Q is open. If, on the other hand, it is closed, the search is

over. Algorithm 17 implements the dominion search procedure $\mathsf{src}_{\mathcal{D}}$. A *compatibility relation* $\succ$ connects the query and the successor operators. The relation holds between states of the partial order and the quasi dominions triples that can be extracted by the query operator. Such a relation defines the domain of the successor operator. The partial order, together with the query and successor operator and the compatibility relation, forms what is called a *dominion space*.

Definition 8.2.2 (Dominion Space)**.** *A* dominion space *for a game* $\mathfrak{D} \in \mathcal{P}$ *is a tuple* $\mathcal{D} \triangleq \langle \mathfrak{D}, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where* (1) $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$ *is a well-founded partial order* w.r.t. $\prec \subset \mathrm{S} \times \mathrm{S}$ *with distinguished element* $\top \in \mathrm{S}$*,* (2) $\succ \subseteq \mathcal{S} \times \mathrm{QD}^{-}_{\mathfrak{D}}$ *is the* compatibility relation*,* (3) $\Re : \mathrm{S} \to \mathrm{QD}_{\mathfrak{D}}$ *is the* query operator *mapping each element* $s \in \mathrm{S}$ *to a quasi dominion triple* $(\mathrm{Q}, \sigma, \wp) \triangleq \Re(s) \in \mathrm{QD}_{\mathfrak{D}}$ *such that, if* $(\mathrm{Q}, \sigma, \wp) \in \mathrm{QD}^{-}_{\mathfrak{D}}$ *then* $s \succ (\mathrm{Q}, \wp, \sigma)$*, and* (4) $\downarrow : \succ \to \mathrm{S}$ *is the* successor operator *mapping each pair* $(s, (\mathrm{Q}, \sigma, \wp)) \in \succ$ *to the element* $s^{\star} \triangleq s \downarrow (\mathrm{Q}, \sigma, \wp) \in \mathrm{S}$ *with* $s^{\star} \prec s$*.*

The notion of dominion space is quite general and can be instantiated in different ways, by providing specific query and successor operators. In [BDM16c], indeed, it is shown that the search procedure $\mathsf{src}_{\mathcal{D}}$ is sound and complete on any dominion space $\mathcal{D}$. In addition, its time complexity is linear in the *execution depth* of the dominion space, namely the length of the longest chain in the underlying partial order compatible with the successor operator, while its space complexity is only logarithmic in the space *size*, since only one state at the time needs to be maintained. A specific instantiation of dominion space, called PP *dominion space*, is the one proposed and studied in [BDM16c]. In the next section, we propose a different one, called RR *dominion space*, which crucially exploits Lemma 8.2.1 in order to prevent a considerable amount of useless reset operations after each quasi dominion composition, to the point that it does not seem obvious whether an exponential lower bound even exists for this new approach.

Algorithm 17: The Searcher.

```
signature src_D : S_D → QD⁺_{𝔇_D}
function src_D(s)
1   (Q, σ, ℘) ← ℜ_D(s)
2   if (Q, σ, ℘) ∈ QD⁺_{𝔇_D} then
3       return (Q, σ, ℘)
    else
4       return src_D(s ↓_D (Q, σ, ℘))
```

8.3 Priority Promotion with Region Recovery

In order to instantiate a dominion space, we need to define a suitable query function to compute quasi dominions and a successor operator to ensure progress in the search for a closed dominion. The priority

promotion algorithm proceeds as follows. The input game is processed in descending order of priority. At each step, a subgame of the entire game, obtained by removing the quasi domains previously computed at higher priorities, is considered. At each priority of parity $\wp$, a quasi $\wp$-domain Q is extracted by the query operator from the current subgame. If Q is closed in the entire game, the search stops and returns Q as result. Otherwise, a successor state in the underlying partial order is computed by the successor operator, depending on whether Q is open in the current subgame or not. In the first case, the quasi $\wp$-dominion is removed from the current subgame and the search restarts on the new subgame that can only contain positions with lower priorities. In the second case, Q is merged together with some previously computed quasi $\wp$-dominion with higher priority. Being a dominion space well-ordered, the search is guaranteed to eventually terminate and return a closed quasi dominion. The procedure requires the solution of two crucial problems: *(a) extracting a quasi dominion* from a subgame and *(b) merging together two quasi $\wp$-dominions* to obtain a bigger, possibly closed, quasi $\wp$-dominion.

Solving problem *(b)* is not trivial, since quasi $\wp$-dominions are not, in general, closed under union. Consider the example in Figure 8.2. Both $\mathrm{Q}_1 = \{\mathtt{a}, \mathtt{c}\}$ and $\mathrm{Q}_2 = \{\mathtt{b}, \mathtt{d}\}$ are quasi 0-dominions. Indeed, $\sigma_1 = \{\mathtt{c} \mapsto \mathtt{c}\}$ and $\sigma_2 = \{\mathtt{d} \mapsto \mathtt{d}\}$ are the corresponding 0-witnesses. However, their union $\mathrm{Q} \triangleq \mathrm{Q}_1 \cup \mathrm{Q}_2$ is not a quasi 0-dominion, since the 1-strategy $\sigma = \{\mathtt{a} \mapsto \mathtt{b}, \mathtt{b} \mapsto \mathtt{a}\}$ forces player 0 to lose along any infinite play starting from either a or b.

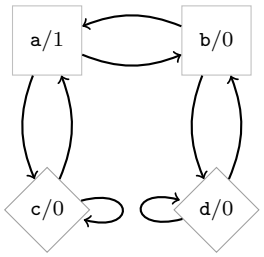


Figure 8.2: Quasi dominions.

A solution to both problems relies on the definition of a specific class of quasi dominions, called *regions*. An $\wp$-region R of a game $\mathcal{D}$ is a special form of quasi $\wp$-dominion of $\mathcal{D}$ with the additional requirement that all the positions in $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$ have the maximal priority $p \triangleq \mathsf{pr}(\mathcal{D}) \equiv_2 \wp$ in $\mathcal{D}$. In this case, we say that $\wp$-region R has priority p. As a consequence, if the opponent $\overline{\wp}$ can escape from the $\wp$-region R, it must visit a position with the highest priority in it, which is of parity $\wp$.

Definition 8.3.1 (Region [BDM16c])**.** *A quasi $\wp$-dominion* R *is an $\wp$-region in $\mathcal{D}$ if* $\mathsf{pr}(\mathcal{D}) \equiv_2 \wp$ *and all the positions in* $\mathsf{esc}^{\overline{\wp}}(\mathrm{R})$ *have priority* $\mathsf{pr}(\mathcal{D})$*,* i.e.*,* $\mathsf{esc}^{\overline{\wp}}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}(\mathsf{pr}(\mathcal{D}))$.

Observe that, in any parity game, an $\wp$-region always exists, for some $\wp \in \{0, 1\}$. In particular, the set of positions of maximal priority in the game always forms an $\wp$-region, with $\wp$ equal to the parity of that maximal priority. In addition, the $\wp$-attractor of an $\wp$-region is always an ($\wp$-maximal) $\wp$-region. A closed $\wp$-region in a game is clearly an $\wp$-dominion in that game. These observations give us an easy and efficient way to extract a quasi dominion from every subgame: collect the $\wp$-attractor of the positions

with maximal priority p in the subgame, where $p \equiv_2 \wp$, and assign p as priority of the resulting region R. This priority, called *measure* of R, intuitively corresponds to an under-approximation of the best priority player $\wp$ can force the opponent $\overline{\wp}$ to visit along any play exiting from R.

Proposition 8.3.1 (Region Extension [BDM16c]). *Let $\partial \in \mathcal{P}$ be a game and $\mathrm{R} \subseteq \mathrm{Ps}$ an $\wp$-region in ∂. Then, $\mathrm{R}^\star \triangleq \mathsf{atr}^\wp(\mathrm{R})$ is an $\wp$-maximal $\wp$-region in ∂.*

A solution to the second problem, the merging operation, is obtained as follows. Given an $\wp$-region R in some game ∂ and an $\wp$-dominion D in a subgame of ∂ that does not contain R itself, the two sets are merged together, if the only moves exiting from $\overline{\wp}$-positions of D in the entire game lead to higher priority $\wp$-regions and R has the lowest priority among them. The priority of R is called the *best escape priority* of D for $\overline{\wp}$. The correctness of this merging operation is established by the following proposition.

Proposition 8.3.2 (Region Merging [BDM16c]). *Let $\partial \in \mathcal{P}$ be a game, $\mathrm{R} \subseteq \mathrm{Ps}$ an $\wp$-region, and $\mathrm{D} \subseteq \mathrm{Ps}_{\partial \setminus \mathrm{R}}$ an $\wp$-dominion in the subgame $\partial \setminus \mathrm{R}$. Then, $\mathrm{R}^\star \triangleq \mathrm{R} \cup \mathrm{D}$ is an $\wp$-region in ∂. Moreover, if both R and D are $\wp$-maximal in ∂ and $\partial \setminus \mathrm{R}$, respectively, then $\mathrm{R}^\star$ is $\wp$-maximal in ∂ as well.*

The merging operation is implemented by promoting all the positions of $\wp$-dominion D to the measure of R, thus improving the measure of D. For this reason, it is called a *priority promotion*. In [BDM16c] it is shown that, after a promotion to some measure p, the regions with measure lower than p might need to be destroyed, by resetting all the contained positions to their original priority. This necessity derives from the fact that the new promoted region may attract positions from lower ones, thereby potentially invalidating their status as regions. Indeed, in some cases, the player that wins by remaining in the region may even change from $\wp$ to $\overline{\wp}$. As a consequence, the reset operation is, in general, unavoidable. The original priority promotion algorithm applies the reset operation to all the lower priority regions. As shown in [BDM16c], the reset operation is the main source of the exponential behaviours of the approach. We shall propose here a different approach that, based on the result of Lemma 8.2.1, can drastically reduce the number of resets needed.

Figure 8.3 illustrates the dominion search procedure on an example game. Diamond shaped positions belong to player 0 and square shaped ones to opponent 1. Each cell of the table contains a computed region. The downward arrow denotes that the region is open in the subgame where is computed, while the upward arrow means that the region gets to be promoted to the priority in the subscript. The measure of the region correspond to the index of the row in which the region is contained. Empty slots in the table represent empty regions, while a slot with symbol = in it means that the it contains the same region as the corresponding slot in the previous column.

Assume the dashed move $(\mathtt{g},\mathtt{k})$ is not present in the game. Then, following the idea sketched above, the first region obtained is the single-position 0-region $\{\mathtt{a}\}$ at priority 8, which is open because of the two moves leading to `e` and `i`. At priority 7, the open 1-region $\{\mathtt{j},\mathtt{k}\}$ is formed, by attracting `k` to `j` according to Proposition 8.3.1, which is open in the subgame where $\{\mathtt{a}\}$ is removed. The procedures proceeds similarly, processing all the priorities down to 1 and extracting the regions reported in the first column of the table of Figure 8.3. Those are all open regions in their corresponding subgames, except for the 1-region $\{\mathtt{g},\mathtt{h}\}$ at priority 1, which is closed in its subgames but not in the entire game. This region has a move $(\mathtt{g},\mathtt{f})$ leading to region 3 and Proposition 8.3.2 is then applied, which promotes this region to 3, obtaining a new 1-region $\{\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h}\}$ with measure 3. This one is again closed in its subgames and, due to move $(\mathtt{h},\mathtt{b})$, triggers another application of Proposition 8.3.2, which promotes all of its positions to region 5 and resets the positions in region 4 to their original priority. The search resumes at priority 5 and the maximization of that region attracts position `c` as well, forming region $\{\mathtt{b},\mathtt{c},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h}\}$ with measure 5. In the resulting subgame, the procedure now extracts the open 1-region $\{\mathtt{d}\}$ at priority 3. The residual game only contains position `i`, that forms a closed 0-region with a move leading to region 8. This triggers a new promotion that resets the position of all the regions with measure lower than 8, namely the regions with measures 7 and 5. After maximization of the target region, positions `b`, `c`, `d`, `h`, and `j` are all attracted to form the 0-region in the first row of column 4. The reset of previous region 7 releases position `k` which now forms an open 0-region of priority 4. Similarly, positions `e` and `f`, reset by the last promotion, form an open 1-region at priority 3. Finally, at priority 1 the closed 1-region $\{\mathtt{g}\}$ is extracted and promoted, by move $(\mathtt{g},\mathtt{f})$, to region 3, forming the set $\{\mathtt{e},\mathtt{f},\mathtt{g}\}$. Since no move from 0-positions lead outside the set, this region is closed in the entire game and a 1-dominion has been found.

During the simulation above, three resets have been performed. The first one resets 0-region $\{\mathtt{c},\mathtt{d}\}$ with measure 4 in column 2, after promotion of region $\{\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h}\}$ to priority 5. Indeed, the maximization of the resulting region $\{\mathtt{b},\mathtt{e},\mathtt{f},\mathtt{g},\mathtt{h}\}$ attracts position `c`, leaving the set $\{\mathtt{d}\}$ with measure 4. However, according to Definition 8.2.1, this is not a quasi 0-dominion, since its 1-escape is empty and player 1 can win by remaining in the set forever. After the promotion of region $\{\mathtt{i}\}$ to 8 in column 3, both the regions in rows 7 and 5 get reset. The maximization of the target region of the promotion, *i.e.*,$\{\mathtt{a},\mathtt{i}\}$, attracts positions `b`, `c`, `d`, `h`, and `j`. As a consequence, for similar reasons described above for position `d`, the residual position `k` at priority 7 must be reset to its original priority. Notice that, in both the considered cases, Lemma 8.2.1 does not apply. Indeed, the 0-witness strategy for region $\{\mathtt{c},\mathtt{d}\}$ is $\sigma = \{\mathtt{d} \mapsto \mathtt{c}\}$, the 0-stay set of the residual region $\{\mathtt{d}\}$ is the set itself, and the restriction of σ to $\{\mathtt{d}\}$ leads outside $\{\mathtt{d}\}$, hence, it does not belong to $\mathrm{Str}^{\mathrm{o}}(\{\mathtt{d}\})$. A similar argument applies to set $\{\mathtt{k}\}$ as well.

As opposed to this case, however, the reset of region 5 can be avoided, thanks to Lemma 8.2.1. Indeed,

a 1-witness for that region is $\sigma = \{\mathtt{e} \mapsto \mathtt{g}, \mathtt{f} \mapsto \mathtt{e}\}$ and, in this case, the residual set after the promotion and the maximization of the target region 8 is $\{\mathtt{e}, \mathtt{f}, \mathtt{g}\}$, whose 1-stay set is $\{\mathtt{e}, \mathtt{f}\}$. The restriction of σ to that set is, however, contained in $\mathrm{Str}^1(\{\mathtt{e}, \mathtt{f}\})$ and the lemma applies. Note that, avoiding the reset of region with measure 5, containing $\{e, f, g\}$ in column 4, would also avoid the computation of regions 4, 3, and 1, and the promotion of region 1 to 3 that leads to column 5. Indeed, the residual region 5 is a 1-region, according to the lemma, and is also closed in the entire game.

If, however, the dashed move $(\mathtt{g}, \mathtt{k})$ was added to the game, the reset of region 5 would be necessary. The reason is that, in this case, the 0-escape set $\{\mathtt{e}, \mathtt{f}, \mathtt{g}\}$ would contain position g, which can escape to position k. As a consequence, $\{\mathtt{e}, \mathtt{f}, \mathtt{g}\}$ would not be a 1-region as the escape set contains a position with priority non-maximal in the subgame, contrary to what is required by Definition 8.3.1.

In summary, we can exploit Lemma 8.2.1 and Definition 8.3.1 to avoid resetting regions after a promotion whenever *(i)* the witness strategy of the residual region satisfies the condition of the lemma, and *(ii)* its escape set only contains positions of maximal priorities in the subgame. This is the core observation that allows the definition of the RR approach, which is formally defined in the following.

	1	2	3	4	5
8	a↓	=	=	a,b,c,d,h,i,j↓	=
7	j,k↓	=	=		
5	b↓	=	b,c,e,f,g,h↓		
4	c,d↓	=		k↓	=
3	e,f↓	e,f,g,h↑5	d↓	e,f↓	e,f,g
1	g,h↑3			g↑3	
0			i↑8		

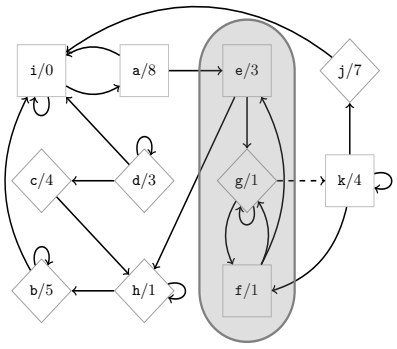


Figure 8.3: Running example.

8.3.1 RR Dominion Space

We can now provide the formal account of the RR dominion space. We shall denote with Rg the set of region triples in $\ni$ and with Rg^- and Rg^+ the sets of open and closed region triples, respectively.

Similarly to the PP algorithm, during the search for a dominion, the computed regions, together with their current measure, are kept track of by means of an auxiliary priority function $\mathsf{r} \in \Delta \triangleq \mathrm{Ps} \to \mathrm{Pr}$, called *region function*. Given a priority $p \in \mathrm{Pr}$, we denote by $\mathsf{r}^{(\geq p)}$ (*resp.*, $\mathsf{r}^{(>p)}$, $\mathsf{r}^{(<p)}$, and $\mathsf{r}^{(\neq p)}$) the

function obtained by restricting the domain of r to the positions with measure greater than or equal to p (*resp.*, greater than, lower than, and different from p). Formally, $\mathsf{r}^{(\sim p)} \triangleq \mathsf{r}{\restriction}\{v \in \mathrm{Ps} : \mathsf{r}(v) \sim p\}$, for $\sim \in \{\geq, >, <, \neq\}$. By $\mathcal{D}_{\mathsf{r}}^{\leq p} \triangleq \mathcal{D} \setminus \mathsf{dom}\big(\mathsf{r}^{(>p)}\big)$, we denote the largest subgame obtained by removing from $\mathcal{D}$ all the positions in the domain of $\mathsf{r}^{(>p)}$. In order for the RR procedure to exploit Lemma 8.2.1, it also needs to keep track of witness strategies of the computed region. To this end, we introduce the notion of witness core. A strategy $\sigma \in \mathrm{Str}^{\wp}(\mathrm{Q})$ is an *$\wp$-witness core* for an $\wp$-region R if *(i)* it is defined on all positions having priority lower than $\mathsf{pr}(\mathrm{R})$, *i.e.*, $\{v \in \mathrm{R} : \mathsf{pr}(v) < \mathsf{pr}(\mathrm{R})\} \subseteq \mathsf{dom}(\sigma)$, and *(ii)* it is a restriction of some $\wp$-witness $\varsigma \in \mathrm{Str}^{\wp}$ for R, *i.e.*, $\sigma \subseteq \varsigma$. Intuitively, a witness core only maintains the essential part of a witness and can be easily transformed into a complete witness by associating every position $v \in \mathsf{stay}^{\wp}(\mathrm{R}) \setminus \mathsf{dom}(\sigma)$ with an arbitrary successor in R. The result of any such completion is an actual witness, since any infinite path passing through v is forced to visit a maximal priority of parity $\wp$.

Definition 8.3.2 (Region-Witness Pair)**.** *Let $\mathsf{r} \in \Delta$ be a priority function, $\tau \in \mathrm{Str}$ a strategy, and $p \in \mathrm{Pr}$ a priority. The pair (r, τ) is a* region-witness pair w.r.t. *p if, for all $q \in \mathsf{rng}(\mathsf{r})$ with $\wp \triangleq q \bmod 2$, $\mathrm{R} \triangleq \mathsf{r}^{-1}(q) \cap \mathrm{Ps}_{\mathcal{D}_{\mathsf{r}}^{\leq q}} \neq \emptyset$, and $\sigma \triangleq \tau{\restriction}\mathrm{R}$, the following two conditions hold:*

1. *if $q \geq p$, then* R *is an $\wp$-region in the subgame $\mathcal{D}_{\mathsf{r}}^{\leq q}$ with $\wp$-witness core σ;*
2. *if $q < p$, there exists a quasi $\wp$-dominion $\mathrm{Q}^{\star} \supseteq \mathrm{R}$ with $\wp$-witness $\sigma^{\star}$ such that* (i) $\mathsf{pr}(\mathrm{Q}^{\star}) = q$, (ii) $\sigma \subseteq \sigma^{\star}$, *and* (iii) $(\mathrm{R} \cap \mathrm{Ps}^{\wp}) \setminus \mathsf{dom}(\sigma) \subseteq \mathsf{pr}^{-1}(q)$.

In addition, r is maximal *above $p \in \mathrm{Pr}$ iff, whenever $q > p$, it holds that* R *is $\wp$-maximal in $\mathcal{D}_{\mathsf{r}}^{\leq q}$ as well.*

As opposed to the PP approach, where a promotion to a priority p resets all the regions of measure lower than p, the RR algorithm resets lower regions only when it cannot ensure their validity. This is done one region at a time, during the descend phase. If, while reading a set $\mathsf{r}^{-1}(q)$ at a certain priority $q < p$, the conditions of Lemma 8.2.1 are not met by $\mathsf{r}^{-1}(q)$ or the escape of that region contains positions of priority lower than q, then $\mathsf{r}^{-1}(q)$ is reset.

Contrary to PP, for which the set contained in r at each measure q must be an $\wp$-region, RR requires such a property only for those regions with measure $q \geq p$, as expressed by Item 1 of the previous definition. For each $q < p$, instead, we simply require the set of positions contained in r at that measure to be a subset of some previously computed quasi dominions of the same player. This is done by requiring that the strategies recorded in τ be subsets of witnesses of these dominions, as described in Item 2. In this way, to verify that $\mathsf{r}^{-1}(q)$ is still a quasi $\wp$-dominion, RR can apply the property stated in Lemma 8.2.1.

The status of the search of a dominion is encoded by the notion of *state* s of the dominion space, which contains the current region-witness pair (r, τ) and the current priority p reached by the search in

$\mathfrak{D}$. Initially, r coincides with the priority function pr of the entire game $\mathfrak{D}$, τ is the empty strategy, and p is set to the maximal priority $\mathsf{pr}(\mathfrak{D})$ available in the game. To each of such states $s \triangleq (\mathsf{r}, \_, p)$, we then associate the *subgame at* s defined as $\mathfrak{D}_s \triangleq \mathfrak{D}_{\mathsf{r}}^{\leq p}$, representing the portion of the original game that still has to be processed.

The following state space specifies the configurations in which the RR procedure can reside and the relative order that the successor function must satisfy.

Definition 8.3.3 (State Space)**.** *A* state space *is a tuple* $\mathcal{S} \triangleq \langle \mathrm{S}, \top, \prec \rangle$*, where:*

1. $\mathrm{S} \subseteq \Delta \times \mathrm{Str} \times \mathrm{Pr}$ *is the set of triples* $s \triangleq (\mathsf{r}, \tau, p)$*, called* states*, where* (a) (r, τ) *is a region-witness pair* w.r.t. p*,* (b) r *is maximal above* p*, and* (c) $p \in \mathsf{rng}(\mathsf{r})$*.*

2. $\top \triangleq (\mathsf{pr}, \varnothing, \mathsf{pr}(\mathfrak{D}))$*;*

3. *for any two states* $s_1 \triangleq (\mathsf{r}_1, \_, p_1), s_2 \triangleq (\mathsf{r}_2, \_, p_2) \in \mathrm{S}$*, it holds that* $s_1 \prec s_2$ *iff either* (a) *there exists a priority* $q \in \mathsf{rng}(\mathsf{r}_1)$ *with* $q \geq p_1$ *such that* (a.i) $\mathsf{r}_1^{(>q)} = \mathsf{r}_2^{(>q)}$ *and* (a.ii) $\mathsf{r}_2^{-1}(q) \subset \mathsf{r}_1^{-1}(q)$*, or* (b) *both* (b.i) $\mathsf{r}_1^{(\geq p_2)} = \mathsf{r}_2^{(\geq p_2)}$ *and* (b.ii) $p_1 < p_2$ *hold.*

Condition 1 requires every region $\mathsf{r}^{-1}(q)$ with measure $q > p$ to be $\wp$-maximal, where $\wp = q \bmod 2$. This implies that $\mathsf{r}^{-1}(q) \subseteq \mathrm{Ps}_{\mathfrak{D}_{\mathsf{r}}^{\leq q}}$. Moreover, the current priority p must be one of the measures recorded in r. Condition 2 specifies the initial state. Finally, Condition 3 defines the ordering relation among states, which the successor operation has to comply with. It asserts that a state s_1 is strictly smaller than another state s_2 if either there is a region recorded in s_1 with some higher measure q that strictly contains the corresponding one in s_2 and all regions with measure grater than q are equal in the two states, or state s_1 is currently processing a lower priority than the one of s_2.

As reported in Definition 8.2.2, the compatibility relation describes which regions are compatibles with a state, *i.e.*, which region triples can be returned by the query operator and used by the successor function. A region triple $(\mathrm{R}, \sigma, \wp)$ is compatible with a state $s \triangleq (\mathsf{r}, \tau, p)$ if R is an $\wp$-region in the current subgame $\mathfrak{D}_s$. Moreover, if such a region is $\wp$-open in that game, it has to be $\wp$-maximal and needs to necessarily contain the current region $\mathsf{r}^{-1}(p)$ of priority p in r.

Definition 8.3.4 (Compatibility Relation)**.** *An open quasi dominion triple* $(\mathrm{R}, \sigma, \wp) \in \mathrm{QD}^{-}$ *is* compatible *with a state* $s \triangleq (\mathsf{r}, \tau, p) \in \mathrm{S}$*, in symbols* $s \succ (\mathrm{R}, \sigma, \wp)$*, iff* (1) $(\mathrm{R}, \sigma, \wp) \in \mathrm{Rg}_{\mathfrak{D}_s}$ *and* (2) *if* R *is* $\wp$*-open in* $\mathfrak{D}_s$ *then* (2.a) R *is* $\wp$*-maximal in* $\mathfrak{D}_s$ *and* (2.b) $\mathsf{r}^{-1}(p) \subseteq \mathrm{R}$*.*

Algorithm 18: Query Function.

```
   signature ℜ: S → 2^Ps × Str × {0,1}
   function ℜ(s)
       let (r, τ, p) = s in
1          ℘ ← p mod 2
2          R* ← r^{-1}(p)
3          (R, σ) ← atr^℘_{∋_s}(R*, τ↾R*)
4      return (R, σ, ℘)
```

Algorithm 18 provides a possible implementation for the query function compatible with the region-recovery mechanism. Given the current state $s \triangleq (\mathsf{r}, \tau, p)$, Line 1 simply computes the parity $\wp$ of the priority p to process at s. Line 3, instead, computes the attractor *w.r.t.* player $\wp$ in subgame $\ni_s$ of the region $\mathrm{R}^\star$ contained in r at p, as determined by Line 2. Observe that here we employ a version of the $\wp$-attractor that, given an $\wp$-witness core for $\mathrm{R}^\star$, also computes the $\wp$-witness for R. This can easily be done by first extending $\tau{\restriction}\mathrm{R}^\star$ with the attraction strategy on the $\wp$-positions in $\mathrm{R} \setminus \mathrm{R}^\star$ and, then, by choosing, for any $\wp$-positions in $\mathrm{R} \setminus \mathsf{dom}(\tau{\restriction}\mathrm{R}^\star)$ with a successor in $\mathrm{R} \setminus \mathrm{R}^\star$, any one of those successors. The resulting set R is, according to Proposition 8.3.1, an $\wp$-maximal $\wp$-region of $\ni_s$ containing $\mathsf{r}^{-1}(p)$ with $\wp$-witness σ.

Algorithm 19: Successor Function.

```
   signature ↓ : ≻ → Δ × Str × Pr
   function s↓(R, σ, ℘)
       let (r, τ, p) = s in
1          if (R, σ, ℘) ∈ Rg^-_{∋_s} then
2              return N(r[R ↦ p], τ ⊎ σ, p)
           else
3              p* ← bep^{℘̄}(R, r)
4              return (r[R ↦ p*], τ ⊎ σ, p*)
```

The promotion operation is based on the notion of best escape priority mentioned above, namely the priority of the lowest $\wp$-region in r that has an incident move coming from the $\wp$-region, closed in the current subgame, that needs to be promoted. This concept is formally defined as follows. Let $I \triangleq Mv \cap ((\mathrm{R} \cap \mathrm{Ps}^{\overline{\wp}}) \times (\mathsf{dom}(\mathsf{r}) \setminus \mathrm{R}))$ be the *interface relation* between R and r, *i.e.*, the set of $\overline{\wp}$-moves exiting from R and reaching some position within a region recorded in r. Then, $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r})$ is set to the minimal measure of those regions that contain positions reachable by a move in I. Formally, $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r}) \triangleq \mathsf{min}(\mathsf{rng}(\mathsf{r}{\restriction}\mathsf{rng}(I)))$. Such a value represents the best priority associated with an $\wp$-region contained in r and reachable by $\overline{\wp}$ when escaping from R. Note that, if R is a closed $\wp$-region in $\ni_s$, then $\mathsf{bep}^{\overline{\wp}}(\mathrm{R}, \mathsf{r})$ is necessarily of parity $\wp$ and greater than the measure p of R. This property immediately follows from the maximality of r above p. Indeed, no move of an $\overline{\wp}$-position can lead to a $\overline{\wp}$-maximal $\overline{\wp}$-region. For instance, for 1-region $\mathrm{R} = \{\mathsf{g}, \mathsf{h}\}$ with measure 1 in Column 1 of Figure 8.3, we have that $I = \{(\mathsf{g}, \mathsf{f}), (\mathsf{h}, \mathsf{b})\}$ and $\mathsf{r}{\restriction}\mathsf{rng}(I) = \{(\mathsf{b}, 5), (\mathsf{f}, 3)\}$. Hence, $\mathsf{bep}^{\mathrm{o}}(\mathrm{R}, \mathsf{r}) = 3$.

Algorithm 19 implements the successor function. Given the state $s \triangleq (\mathsf{r}, \tau, p)$ and one of its possible compatible region triples $(\mathrm{R}, \sigma, \wp)$ open in the original game $\ni$, it produces a successor state $s^\star \prec s$. Line 1

checks if R is open in the subgame ∂_s as well. If this is the case, at Line 2, the next state $s^\star$ is generated by the auxiliary function, called *next state function*, described below, which also applies the required resets. On the other hand, if R is closed in ∂_s, the procedure performs the promotion of R, by exploiting Proposition 8.3.2. Indeed, Line 3 computes the best escape priority $p^\star$ to which R needs to be promoted, while Line 4 sets the measure of R to $p^\star$ and merges the strategies contained in τ with the $\wp$-witness σ of R. Observe that, unlike in the PP successor function, no reset operation is applied to r at this stage.

Finally, Algorithm 20 reports the pseudo code of the next state function, the essential core of the RR approach. At a state $s \triangleq (\mathsf{r}, \sigma, p)$, Line 1 computes the priority $p^\star$ of the successive set of positions $\mathsf{r}^{-1}(p^\star)$ occurring in r starting from p in descending order of priority. Then, Line 2 verifies whether this set is actually a region, by computing the truth value of the formula ϕ described below applied to the triple $(\mathsf{r}, \tau, p^\star)$. If this is the case, the successor state $(\mathsf{r}, \tau, p^\star)$ of s is returned at Line 3. On the other hand, if the check fails, the algorithm resets, at Line 4, the positions in $\mathsf{r}^{-1}(p^\star)$ to their original priority stored in the priority function pr of the game, and deletes, at Line 5, the associated strategy contained in τ. Finally, at Line 6, the next state function is recursively applied to the newly obtained state.

Algorithm 20: Next State Function.

```
   signature N : S → Δ × Str × Pr
   function N(s)
       let (r, τ, p) = s in
1          p⋆ ← max(rng(r^(<p)))
2          if φ(r, τ, p⋆) then
3              return (r, τ, p⋆)
           else
4              r⋆ ← pr ⊎ r^(≠p⋆)
5              τ⋆ ← σ \ r^{-1}(p⋆)
6              return N(r⋆, τ⋆, p)
```

To check whether a set of positions $\mathrm{R} \triangleq \mathsf{r}^{-1}(p)$ at a certain priority $q < p$ is an $\wp$-region with $\wp \triangleq q \bmod 2$, we make use of the formula $\phi(\mathsf{r}, \tau, q) \triangleq \phi_i(\mathsf{r}, \tau, q) \wedge \phi_{ii}(\mathsf{r}, q)$, which verifies that *(i)* $\sigma \triangleq \tau{\restriction}\mathrm{R}$ is a witness core for R and *(ii)* the escape only contains positions of maximal priorities in the subgame. The two predicates are formally defined as follows.

$$\begin{aligned} \phi_i(\mathsf{r}, \tau, q) &\triangleq \forall v \in \mathrm{Ps}^{\wp}_{\partial^\star} \cap \mathsf{dom}(\sigma)\,.\,\sigma(v) \in \mathrm{R} \\ \phi_{ii}(\mathsf{r}, q) &\triangleq \mathsf{esc}^{\overline{\wp}}_{\partial^\star}(\mathrm{R}) \subseteq \mathsf{pr}^{-1}_{\partial^\star}(\mathsf{pr}(\partial^\star)) \end{aligned} \quad \text{with} \quad \begin{cases} \partial^\star \triangleq \partial^{\leq q}_{\mathsf{r}},\ \wp \triangleq q \bmod 2, \\ \mathrm{R} \triangleq \mathsf{r}^{-1}(q),\ \sigma \triangleq \tau{\restriction}\mathrm{R}. \end{cases}$$

Intuitively, if $\phi_i(\mathsf{r}, \tau, q)$ holds, we are sure that $\sigma \in \mathrm{Str}^{\wp}(\mathrm{R})$. Moreover, due to Item 2 of Definition 8.3.2, R is a subset of a quasi $\wp$-dominion having a witness containing σ. Therefore, by Lemma 8.2.1, we immediately derives that σ is a witness core for R. Additionally, the formula $\phi_{ii}(\mathsf{r}, q)$ just checks that the second condition of the definition of region is also met.

The following theorem establishes the correctness of the RR approach.

Theorem 8.3.1 (Dominion Space)**.** *For a game* $\ni$*, the* structure $\mathcal{D} \triangleq \langle \ni, \mathcal{S}, \succ, \Re, \downarrow \rangle$*, where* $\mathcal{S}$ *is given in Definition 8.3.3,* $\succ$ *is the relation of Definition 8.3.4, and* $\Re$ *and* $\downarrow$ *are the functions computed by Algorithms 18 and 19 is a dominion space.*

The RR procedure drastically reduces the number of resets needed to solve a game *w.r.t.* PP. In particular, the exponential worst-case game presented in [BDM16c] does not work any more, since the execution depth of the associated RR dominion space is only quadratic in the parameter of game family. However, recently an exponential worst-case has been discovered and presented in [vD19]. Moreover, at the present time, we are not able to provide a better asymptotic upper bound for the time complexity *w.r.t.* the PP one.

8.4 Experimental Evaluation

The technique proposed in the paper has been implemented in the tool PGSolver [FL09], which collects implementations of several parity game solvers proposed in the literature and provides benchmarking tools that can be used to evaluate the solver performances.[1]

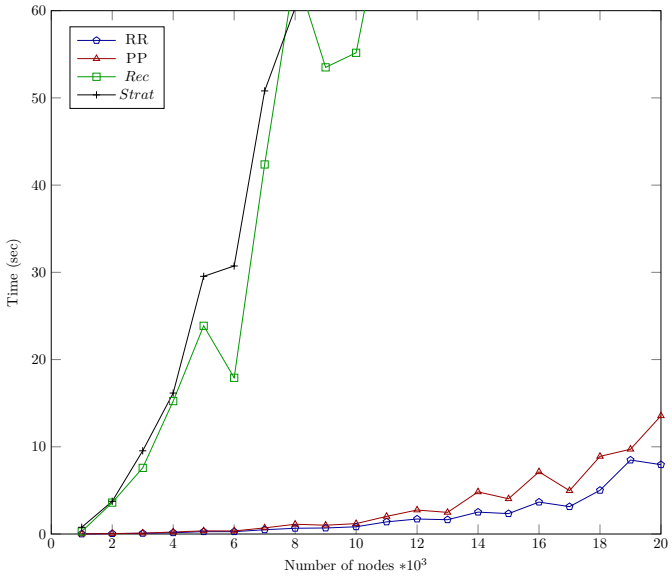


Figure 8.4: Comparative results on 2000 random games with up to 20000 positions (from [BDM16c]).

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

Figure 8.4 compares the running times of the new algorithm RR against the original version PP and the well-known solvers *Rec* and *Str*, implementing the recursive algorithm [Zie98] and the strategy improvement technique [VJ00], respectively. This first pool of benchmarks is taken from [BDM16c] and involves 2000 random games of size ranging from 1000 to 20000 positions and 2 outgoing moves per position. Interestingly, random games with very few moves prove to be much more challenging for the priority promotion based approaches than those with a higher number of moves per position, and often require a much higher number of promotions. Since the behaviour of the solvers is typically highly variable, even on games of the same size and priorities, to summarise the results we took the average running time on clusters of games.

Therefore, each point in the graph shows the average time over a cluster of 100 different games of the same size: for each size value n, we chose the numbers $k = n \cdot i/10$ of priorities, with $i \in [1, 10]$, and 10 random games were generated for each pair n and k. We set a time-out to 180 seconds (3 minutes). The new solver RR shows a significant improvement on all the benchmarks. All the other solvers provided in PGSolver, including the Dominion Decomposition [JPZ08] and the Big Step [Sch07] algorithms, perform quite poorly on those games, hitting the time-out already for very small instances. Figure 8.4 shows only the best performing ones on the considered games, namely *Rec* and *Str*.

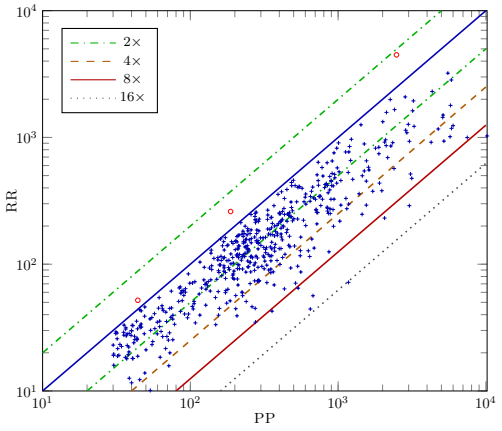


Figure 8.5: Comparison between PP and RR on random games with 50000 positions on a logarithmic scale.

Similar experiments were also conducted on random games with a higher number of moves per position and up to 100000 positions. The resulting games turn out to be very easy to solve by all the priority promotion based approaches. The reason seems to be that the higher number of moves significantly increases the dimension of the computed regions and, consequently, also the chances to find a closed one. Indeed, the number of promotions required by PP and RR on all those games is

typically zero, and the whole solution time is due exclusively to a very limited number of attractors needed to compute the few regions contained in the games. We reserve the presentation of the results for the extended version.

To further stress the RR technique in comparison with PP, we also generated a second pool of much harder benchmarks, containing more than 500 games, each with 50000 positions, 12000 priorities and 2 moves per positions. We selected as benchmarks only random games whose solution requires PP between 30 and 6000 seconds. The results comparing PP and RR are reported in Figure 8.5 on a logarithmic scale. The figure shows that in three cases PP performs better than RR. This is due to the fact that the two algorithms may follow different solution paths within the dominion space and that following the new technique may, in some cases, defer the discovery of a closed dominion. Nonetheless, the RR algorithm does pay off significantly on the vast majority of the benchmarks, often solving a game between two to sixteen times faster than PP.

In [BDM16c] it is shown that PP solves all the known exponential worst cases for the other solvers without promotions and, clearly, the same holds of RR as well. As a consequence, RR only requires polynomial time on those games and the experimental results coincide with the ones for PP.

Chapter 9

Solving Mean-Payoff Games via Quasi Dominions

In this final chapter we show how the notion of quasi dominion used in the Priority Promotion approaches can be adapted to another type of games, specifically mean-payoff games, where the concept of promotion of a priority is lifted to promotion of a payoff. We conjecture that such a lifting can be possibly applicable to other closely related games such as discounted mean-payoff and stochastic games.

This work has been presented at the

- Highlights of Logic, Games and Automata (HIGHLIGHTS 2019), held in September 15-18, 2019, in Warsaw, Poland, with the title of "Solving Mean-Payoff Games via Quasi Dominions".

The content of the chapter is based on [BDM19b].

9.1 Abstract

We propose a novel mean-payoff progress measure approach that enriches measures with the notion of *quasi dominions*, originally introduced in [BDM16c] for parity games. These are sets of positions with the property that as long as the opponent chooses to play to remain in the set, it loses the game for sure, hence its best choice is always to try to escape. A quasi dominion from where is not possible escaping is a winning set for the other player. Progress measure approaches, such as the one of [BCD$^+$11], typically focus on finding the best choices of the opponent and little information is gathered on the

other player. In this sense, they are intrinsically asymmetric. Enriching the approach with quasi dominions can be viewed as a way to also encode the best choices of the player, information that can be exploited to speed up convergence significantly. The main difficulty here is that suitable lift operators in the new setting do not enjoy monotonicity. Such a property makes proving completeness of classic progress measure approaches almost straightforward, as monotonic operators do admit a least fixpoint. Instead, the lift operator we propose is only inflationary (specifically, non-decreasing) and, while still admitting fixpoints [Bou49, Wit50], need not have a least one. Hence, providing a complete solution algorithm proves more challenging. The advantages, however, are significant. On the one hand, the new algorithm still enjoys the same worst-case complexity of the best known algorithm for the problem proposed in [BCD$^+$11]. On the other hand, we show that there exist families of games on which the classic approach requires a number of operations that can be made arbitrarily larger than the one required by the new approach. Experimental results also witness the fact that this phenomenon is by no means isolated, as the new algorithm performs orders of magnitude better than the algorithm developed in [BCD$^+$11].

9.2 Solving Mean-Payoff Games via Progress Measures

The abstract notion of progress measure [Kla91] has been introduced as a way to encode global properties on paths of a graph by means of simpler local properties of adjacent vertexes. In the context of MPGs, the graph property of interest, called *mean-payoff property*, requires that the mean payoff of every infinite path in the graph be non-positive. More precisely, in game theoretic terms, a *mean-payoff progress measure* witnesses the existence of strategy σ_1 for player 1 such that each path in the graph induced by fixing that strategy on the arena satisfies the desired property. A mean-payoff progress measure associates with each vertex of the underlying graph a value, called *measure*, taken from the set of extended natural numbers $\mathsf{N}_\infty \triangleq \mathbb{N} \cup \{\infty\}$, endowed with an ordering relation $\leq$ and an addition operation $+$, which extend the standard ordering and addition over the naturals in the usual way. Measures are associated with positions in the game and the measure of a position v can intuitively be interpreted as an estimate of the payoff that player 0 can enforce on the plays starting in v. In this sense, they measure "how far" v is from satisfying the mean-payoff property, with the maximal measure ∞ denoting failure of the property for v. More precisely, the 1-strategy induced by a progress measure ensures that measures do not increase along the paths of the induced graph. This, in turn, ensures that every path eventually gets trapped in a non-positive-weight cycle, thereby witnessing a win for player 1.

To obtain a progress measure, one starts from some suitable association of position of the game with

measures. The local information encoded by these measures is then propagated back along the edges of the underlying graph so as to associate with each position the information gathered along plays of some finite length starting from that position. The propagation process is performed according to the following intuition. The measures of positions adjacent to v are propagated back to v only if those measures push v further away from the property. This propagation is achieved by means of a measure stretch operation $+$, which adds, when appropriate, the measure of an adjacent position to the weight of a given position. This is established by comparing the measure of v with those of its adjacent positions, since, for each position v, the mean-payoff property is defined in terms of the sum of the weights encountered along the plays from that position. The process ends when no position can be pushed further away from the property and each position is not dominated by any, respectively one, of its adjacents, depending on whether that position belongs to player 0 or to player 1, respectively. The positions that did not reach measure ∞ are those from which player 1 can win game and the set of measures currently associated with such positions forms a mean-payoff progress measure for the game.

To make the above intuitions precise, we introduce the notion of measure function, progress measure, and an algorithm for computing progress measures correctly. It is worth noticing that the progress-measure based approach as described in [BCD+11], called SEPM from now on, can be easily recast equivalently in the form below. A *measure function* $\mu\colon \mathrm{Ps} \to \mathsf{N}_\infty$ maps each position v in the game to a suitable measure $\mu(v)$. The order $\leq$ of the measures naturally induces a pointwise partial order $\sqsubseteq$ on the measure functions defined in the usual way, namely, for any two measure functions μ_1 and μ_2, we write $\eta_1 \sqsubseteq \eta_2$ if $\mu_1(v) \leq \mu_2(v)$, for all positions v. The set of measure functions over a measure space, together with the induced ordering $\sqsubseteq$, forms a *measure-function space.*

Definition 9.2.1 (Measure-Function Space)**.** *The* measure-function space *is the* partial order $\mathcal{F} \triangleq \langle \mathrm{MF}, \sqsubseteq \rangle$ *whose components are defined as reported in the following:*

1. $\mathrm{MF} \triangleq \mathrm{Ps} \to \mathsf{N}_\infty$ *is the set of all functions* $\mu \in \mathrm{MF}$*, called* measure functions*, mapping each position* $v \in \mathrm{Ps}$ *to a measure* $\mu(v) \in \mathsf{N}_\infty$*;*
2. *for all* $\mu_1, \mu_2 \in \mathrm{MF}$*, it holds that* $\mu_1 \sqsubseteq \mu_2$ *if* $\mu_1(v) \leq \mu_2(v)$*, for all positions* $v \in \mathrm{Ps}$*.*

The 0-denotation *(*resp.*,* 1-denotation*) of a measure function* $\mu \in \mathrm{MF}$ *is the set* $\|\mu\|_0 \triangleq \mu^{-1}(\infty)$ *(*resp.*,* $\|\mu\|_1 \triangleq \overline{\mu^{-1}(\infty)}$*) of all positions having maximal (*resp.*, non-maximal) measure associated within* μ*.*

Assuming that a given position v has an adjacent with measure η, a measure update of η *w.r.t.* v is obtained by the stretch operator $+\colon \mathsf{N}_\infty \times \mathrm{Ps} \to \mathsf{N}_\infty$, defined as

$$\eta + v \triangleq \mathsf{max}\{0, \eta + \mathsf{wg}(v)\},$$

which corresponds to the payoff estimate that the given position will obtain by choosing to follow the move leading to the prescribed adjacent.

A *mean-payoff progress measure* is such that the measure associated with each game position v needs not be increased further in order to beat the actual payoff of the plays starting from v. In particular, it can be defined by taking into account the opposite attitude of the two players in the game. While the player 0 tries to push toward higher measures, the player 1 will try to keep the measures as low as possible. A measure function in which the payoff of each 0-position (*resp.*, 1-position) v is not dominated by the payoff of all (*resp.*, some of) its adjacents augmented with the weight of v itself meets the requirements.

Definition 9.2.2 (Progress Measure)**.** *A measure function* $\mu \in \mathrm{MF}$ *is a* progress measure *if the following two conditions hold true, for all positions* $v \in \mathrm{Ps}$*:*

1. $\mu(u) + v \leq \mu(v)$*, for all adjacents* $u \in Mv(v)$ *of* v*, if* $v \in \mathrm{Ps}_0$*;*
2. $\mu(u) + v \leq \mu(v)$*, for some adjacent* $u \in Mv(v)$ *of* v*, if* $v \in \mathrm{Ps}_1$*.*

The following theorem states the fundamental property of progress measures, namely, that every position associated with a non-maximal value is won by player 1.

Theorem 9.2.1 (Progress Measure)**.** *Let* $\mu \in \mathrm{MF}$ *be a progress measure. Then,* $\|\mu\|_1 \subseteq \mathrm{Wn}_1$*.*

Proof. Consider a $\boxminus$-strategy $\sigma_1 \in \mathrm{Str}_1$ for which all measures $\mu(v)$ of positions $v \in \|\mu\|_1 \cap \mathrm{Ps}_1$ are a progress at v *w.r.t.* the measures $\mu(\sigma_1(v))$ of their adjacents $\sigma_1(v)$, formally, $\mu(\sigma_1(v)) + v \leq \mu(v)$. The existence of such a strategy is ensured by the fact that μ is a progress measure. Indeed, by Condition 2 of Definition 9.2.2, there necessarily exists a adjacent $u^\star \in Mv(v)$ of v such that $\mu(u^\star) + v \leq \mu(v)$. Now, it can be shown that σ_1 is a winning strategy for player 1 from all the positions in $\|\mu\|_1$, which implies that $\|\mu\|_1 \subseteq \mathrm{Wn}_1$. To do this, let us consider a 0-strategy $\sigma_0 \in \mathrm{Str}_0$ and the associated play $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$ starting at a position $v \in \|\mu\|_1$. Assume, by contradiction, that π is won by player 0. Since the game $\supset$ is finite, π must contain a finite simple cycle, and so a finite simple path, with strictly positive total weight sum. In other words, there exist two natural numbers $h \in \mathbb{N}$ and $k \in \mathbb{N}_+$ such that $(\pi)_h = (\pi)_{h+k}$ and $\mathsf{wg}(\rho) = \sum_{i=h}^{h+k-1} \mathsf{wg}((\pi)_i) > 0$, where $\rho \triangleq ((\pi)_{\geq h})_{<h+k}$ is the simple path named above. Now, recall that, $((\pi)_i, (\pi)_{i+1}) \in Mv$, for all indexes $i \in \mathbb{N}$. Thus, by both conditions of Definition 9.2.2, and the notion of play, we have that

$$\mu((\pi)_{i+1}) + (\pi)_i \leq \mu((\pi)_i).$$

Via a trivial induction, it is immediate to see that $\mu((\pi)_i) \leq S$, where $S \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathrm{Ps} \wedge \mathsf{wg}(v) > 0\} < \infty$, for all $i \in \mathbb{N}$, since $\mu((\pi)_0) = \mu(v) \neq \infty$, being $v \in \|\mu\|_1$. As a consequence, due

to the definition of the measure stretch operator, it holds that

$$\mu((\pi)_{i+1}) + \mathsf{wg}((\pi)_i) \leq \mu((\pi)_i) \leq S.$$

Hence, by summing together all the inequalities having indexes $i \in \mathbb{N}$ with $h \leq i < h + k$, we obtain

$$\sum_{i=h+1}^{h+k} \mathsf{pf}_\mu((\pi)_i) + \sum_{i=h}^{h+k-1} \mathsf{wg}((\pi)_i) \leq \sum_{i=h}^{h+k-1} \mathsf{pf}_\mu((\pi)_i) < \infty,$$

which simplifies in $\mathsf{wg}(\rho) = \sum_{i=h}^{h+k-1} \mathsf{wg}((\pi)_i) \leq 0$, since $\mathsf{pf}_\mu((\pi)_{h+k}) = \mathsf{pf}_\mu((\pi)_h)$. However, this contradicts the above assumption $\mathsf{wg}(\rho) > 0$. Therefore, σ_1 is a winning strategy for player 1 on $\|\mu\|_1$ as required by the theorem statement. □

In order to obtain a progress measure from a given measure function, one can iteratively adjust the current measure values in such a way to force the progress condition above among adjacent positions. To this end, we define the *lift operator* $\mathsf{lift}\colon \mathrm{MF} \to \mathrm{MF}$ as follows:

$$\mathsf{lift}(\mu)(v) \triangleq \begin{cases} \mathsf{max}\{\mu(w) + v : w \in Mv(v)\}, & \text{if } v \in \mathrm{Ps}_0; \\ \mathsf{min}\{\mu(w) + v : w \in Mv(v)\}, & \text{otherwise.} \end{cases}$$

Note that the lift operator is clearly monotone and, therefore, admits a least fixpoint. A mean-payoff progress measure can, then, be obtained by repeatedly applying this operator until a fixpoint is reached, starting from the minimal measure function $\mu_0 \triangleq \{v \in \mathrm{Ps} \mapsto 0\}$ that assigns measure 0 to all the positions in the game. The following *solver operator* applied to μ_0 computes the desired solution:

$$\mathsf{sol} \triangleq \mathsf{lfp}\, \mu \,.\, \mathsf{lift}(\mu)\colon \mathrm{MF} \to \mathrm{MF}.$$

Observe that the measures generated by the procedure outlined above have a fairly natural interpretation. Each positive measure, indeed, under-approximates the weight that player 0 can enforce along finite prefixes of the plays from the corresponding positions. This follows from the fact that, while player 0 maximizes its measures along the outgoing moves, player 1 minimizes them. In this sense, each positive measure witnesses the existence of a positively-weighted finite prefix of a play that player 0 can enforce. Let $S \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathrm{Ps} \wedge \mathsf{wg}(v) > 0\}$ be the sum of all the positive weights in the game. Clearly, the maximal payoff of a simple play in the underlying graph cannot exceed S. Therefore, a measure greater than S witnesses the existence of a cycle whose payoff diverges to infinity and is won, thus, by player 0. Hence, any measure strictly greater than S can be substituted with the value ∞. This observation established the termination of the algorithm and is instrumental to its completeness

proof. Indeed, at the fixpoint, the measures actually coincide with the highest payoff player 0 is able to guarantee. Soundness and completeness of the above procedure have been established in [BCD+11], where the authors also show that, despite the algorithm requiring $\mathrm{O}(n \cdot S) = \mathrm{O}\big(n^2 \cdot W\big)$ lift operations in the worst-case, with n the number of positions and W the maximal positive weight in the game, the overall cost of these lift operations is $\mathrm{O}(S \cdot m \cdot \log S) = \mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W))$, with m the number of moves and $\mathrm{O}(\log S)$ the cost of each arithmetic operation necessary to compute the stretch of the measures.

9.3 Solving Mean-Payoff Games via Quasi Dominions

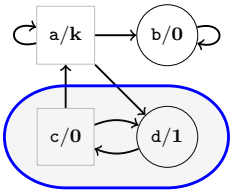


Figure 9.1: An MPG.

Let us consider the simple example game depicted in Figure 9.1, where the shape of each position indicates the owner, circles for player 0 and square for its opponent 1, and, in each label of the form ℓ/w, the letter w corresponds to the associated weight, where we assume $k > 1$. Starting from the smallest measure function $\mu_0 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d} \mapsto 0\}$, the first application of the lift operator returns $\mu_1 = \{\mathtt{a} \mapsto k; \mathtt{b}, \mathtt{c} \mapsto 0; \mathtt{d} \mapsto 1\} = \mathsf{lift}(\mu_0)$. After that step, the following iterations of the fixpoint alternatively updates positions $\mathtt{c}$ and $\mathtt{d}$, since the other ones already satisfy the progress condition. Being $\mathtt{c} \in \mathrm{Ps}_1$, the lift operator chooses for it the measure computed along the move $(\mathtt{c}, \mathtt{d})$, thus obtaining $\mu_2(\mathtt{c}) = \mathsf{lift}(\mu_1)(\mathtt{c}) = \mu_1(\mathtt{d}) = 1$. Subsequently, $\mathtt{d}$ is updated to $\mu_3(\mathtt{d}) = \mathsf{lift}(\mu_2)(\mathtt{d}) = \mu_2(\mathtt{c}) + 1 = 2$. A progress measure is obtained after exactly $2k + 1$ iterations, when the measure of $\mathtt{c}$ reaches value k and $\mathtt{d}$ value $k + 1$. Note, however, that the choice of the move $(\mathtt{c}, \mathtt{d})$ is clearly a losing strategy for player 1, as remaining in the highlighted region would make the payoff from position $\mathtt{c}$ diverge. Therefore, the only reasonable choice for player 1 is to exit from that region by taking the move leading to position $\mathtt{a}$. An operator able to diagnose this phenomenon early on could immediately discard the move $(\mathtt{c}, \mathtt{d})$ and jump directly to the correct payoff obtained by choosing the move to position $\mathtt{a}$. As we shall see, such an operator might lose the monotonicity property and recovering the completeness of the resulting approach will prove more involved.

In the rest of this article we shall devise a progress operator that does precisely that. To this end, we start by providing a notion of *quasi dominion*, originally introduced for parity games in [BDM16c], which can be exploited in the context of MPGs.

Definition 9.3.1 (Quasi Dominion)**.** *An arbitrary set of positions* $\mathrm{Q} \subseteq \mathrm{Ps}$ *is a* quasi 0-dominion *if there exists a* 0*-strategy* $\sigma_0 \in \mathrm{Str}_0(\mathrm{Q})$*, called* 0-witness for Q*, such that, for all* 1*-strategies* $\sigma_1 \in \mathrm{Str}_1(\mathrm{Q})$ *and positions* $v \in \mathrm{Q}$*, the induced play* $\pi = \mathsf{play}((\sigma_0, \sigma_1), v)$*, called* (σ_0, v)-play in Q*, satisfies* $\mathsf{wg}(\pi) > 0$*. If the condition* $\mathsf{wg}(\pi) > 0$ *holds only for infinite plays* π*, then* Q *is called* weak quasi 0-dominion*.*

Essentially, a quasi 0-dominion consists in a set Q of positions starting from which player 0 can force plays in Q of positive weight. Analogously, any infinite play that player 0 can force in a weak quasi 0-dominion has positive weight. Clearly, any quasi 0-dominion is also a weak quasi 0-dominion. Moreover, the latter are closed under subsets, while the former are not. It is an immediate consequence of the definition above that all infinite plays induced by the 0-witness, if any, necessarily have infinite weight and, thus, are winning for player 0. Indeed, every such a play π is regular, *i.e.* it can be decomposed

into a prefix π' and a simple cycle $(\pi'')^\omega$, *i.e.* $\pi = \pi'(\pi'')^\omega$, since the strategies we are considering are memoryless. Now, $\mathsf{wg}((\pi'')^\omega) > 0$, so, $\mathsf{wg}(\pi'') > 0$, which implies $\mathsf{wg}((\pi'')^\omega) = \infty$. Hence, $\mathsf{wg}(\pi) = \infty$.

Proposition 9.3.1. *Let* Q *be a weak quasi* 0*-dominion with* $\sigma_0 \in \mathrm{Str}_0(\mathrm{Q})$ *one of its* 0*-witnesses and* $\mathrm{Q}^\star \subseteq \mathrm{Q}$*. Then, for all* 1*-strategies* $\sigma_1 \in \mathrm{Str}_1(\mathrm{Q}^\star)$ *and positions* $v \in \mathrm{Q}^\star$ *the following holds: if the* $(\sigma_{0\restriction \mathrm{Q}^\star}, v)$*-play* $\pi = \mathsf{play}((\sigma_{0\restriction \mathrm{Q}^\star}, \sigma_1), v)$ *is infinite, then* $\mathsf{wg}(\pi) = \infty$*.*

From Proposition 9.3.1, it directly follows that, if a weak quasi 0-dominion Q is *closed w.r.t.* its 0*-witness*, namely all the induced plays are infinite, then it is a 0-dominion, hence is contained in Wn_0.

Consider again the example of Figure 9.1. The set of position $\mathrm{Q} \triangleq \{\mathtt{a}, \mathtt{c}, \mathtt{d}\}$ forms a quasi 0-dominion whose 0-witness is the only possible 0-strategy mapping position `d` to `c`. Indeed, any infinite play remaining in Q forever and compatible with that strategy (*e.g.*, the play from position `c` when player 1 chooses the move from `c` leading to `d` or the one from `a` to itself or the one from `a` to `d`) grants an infinite payoff. Any finite compatible play, instead, ends in position `a` (*e.g.*, the play from `c` when player 1 chooses the move from `c` to `a` and then one from `a` to `b`) giving a payoff of at least $k > 0$. On the other hand, $\mathrm{Q}^\star \triangleq \{\mathtt{c}, \mathtt{d}\}$ is only a weak quasi 0-dominion, as player 1 can force a play of weight 0 from position `c`, by choosing the exiting move $(\mathtt{c}, \mathtt{a})$. However, the internal move $(\mathtt{c}, \mathtt{d})$ would lead to an infinite play in $\mathrm{Q}^\star$ of infinite weight.

The crucial observation here is that the best choice for player 1 in any position of a (weak) quasi 0-dominion is to exit from it as soon as it can, while the best choice for player 0 is to remain inside it as long as possible. The idea of the algorithm we propose in this section is to precisely exploit the information provided by the quasi dominions in the following way. Consider the example above. In position `a` player 1 must choose to exit from $\mathrm{Q} = \{\mathtt{a}, \mathtt{c}, \mathtt{d}\}$, by taking the move $(\mathtt{a}, \mathtt{b})$, without changing its measure, which would corresponds to its weight k. On the other hand, the best choice for player 1 in position `c` is to exit from the weak quasi-dominion $\mathrm{Q}^\star = \{\mathtt{c}, \mathtt{d}\}$, by choosing the move $(\mathtt{c}, \mathtt{a})$ and lifting its measure from 0 to k. Note that this contrasts with the minimal measure-increase policy for player 1 employed in [BCD+11], which would keep choosing to leave `c` in the quasi-dominion by following the move to `d`, which gives the minimal increase in measure of value 1. Once `c` is out of the quasi-dominion, though, the only possible move for player 0 is to follow `c`, taking measure $k + 1$. The resulting measure function is a progress measure and the solution has, thus, been reached.

In order to make this intuitive idea precise, we need to be able to identify quasi dominions first. Interestingly enough, the measure functions μ defined in the previous section do allow to identify a quasi dominion, namely the set of positions $\overline{\mu^{-1}(0)}$ having positive measure. Indeed, as observed at the end of that section, a positive measure witnesses the existence of a positively-weighted finite play that player 0

can enforce from that position onward, which is precisely the requirement of Definition 9.3.1. In the example of Figure 9.1, $\overline{\mu_0^{-1}(0)} = \emptyset$ and $\overline{\mu_1^{-1}(0)} = \{\mathtt{a}, \mathtt{c}, \mathtt{d}\}$ are both quasi dominions, the first one *w.r.t.* the empty 0-witness and the second one *w.r.t.* the 0-witness $\sigma_0(\mathtt{d}) = \mathtt{c}$.

We shall keep the quasi-dominion information in pairs (μ, σ), called *quasi-dominion representations* (QDR, for short), composed of a measure function μ and a 0-strategy σ, which corresponds to one of the 0-witnesses of the set of positions with positive measure in μ. The connection between these two components is formalized in the definition below that also provides the partial order over which the new algorithm operates.

Definition 9.3.2 (QDRSpace)**.** *The* quasi-dominion-representation space *is the* partial order $\mathcal{Q} \triangleq \langle \mathrm{QDR}, \sqsubseteq \rangle$*, whose components are defined as prescribed in the following:*

1. $\mathrm{QDR} \subseteq \mathrm{MF} \times \mathrm{Str}_0$ *is the set of all pairs* $\varrho \triangleq (\mu_\varrho, \sigma_\varrho) \in \mathrm{QDR}$*, called* quasi-dominion-representations*, composed of a measure function* $\mu_\varrho \in \mathrm{MF}$ *and a* 0*-strategy* $\sigma_\varrho \in \mathrm{Str}_0(\mathsf{Q}(\varrho))$*, where* $\mathsf{Q}(\varrho) \triangleq \overline{\mu_\varrho^{-1}(0)}$*, for which the following four conditions hold:*
   (a) $\mathsf{Q}(\varrho)$ *is a quasi* 0*-dominion enjoying* σ_ϱ *as a* 0*-witness;*
   (b) $\|\mu_\varrho\|_0$ *is a* 0*-dominion;*
   (c) $\mu_\varrho(v) \leq \mu_\varrho(\sigma_\varrho(v)) + v$*, for all* 0*-positions* $v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_0$*;*
   (d) $\mu_\varrho(v) \leq \mu_\varrho(u) + v$*, for all* 1*-positions* $v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_1$ *and adjacents* $u \in Mv(v)$*;*
2. *for all* $\varrho_1, \varrho_2 \in \mathrm{QDR}$*, it holds that* $\varrho_1 \sqsubseteq \varrho_2$ *if* $\mu_{\varrho_1} \sqsubseteq \mu_{\varrho_2}$ *and* $\sigma_{\varrho_1}(v) = \sigma_{\varrho_2}(v)$*, for all* 0*-positions* $v \in \mathsf{Q}(\varrho_1) \cap \mathrm{Ps}_0$ *with* $\mu_{\varrho_1}(v) = \mu_{\varrho_2}(v)$*.*

The α-denotation $\|\varrho\|_\alpha$ *of a* QDRϱ*, with* $\alpha \in \{0, 1\}$*, is the* α*-denotation* $\|\mu_\varrho\|_\alpha$ *of its measure function.*

Condition 1a is obvious. Condition 1b, instead, requires that every position with infinite measure is indeed won by player 0 and is crucial to guarantee the completeness of the algorithm. Finally, Conditions 1c and 1d ensure that every positive measure under approximates the actual weight of some finite play within the induced quasi dominion. This is formally captured by the following proposition, which can be easily proved by induction on the length of the play.

Proposition 9.3.2. *Let* ϱ *be a* QDR*and* $v\pi u$ *a finite path starting at position* $v \in \mathrm{Ps}$ *and terminating in position* $u \in \mathrm{Ps}$ *compatible with the* 0*-strategy* σ_ϱ*. Then,* $\mu_\varrho(v) \leq \mathsf{wg}(v\pi) + \mu_\varrho(u)$*.*

It is immediate to see that every MPG admits a non-trivial QDRspace, since the pair (μ_0, σ_0), with μ_0 the smallest measure function and σ_0 the empty strategy, trivially satisfies all the required conditions.

Proposition 9.3.3. *Every* MPG *has a non-empty* QDR *space associated with it.*

The solution procedure we propose, called QDPM from now on, can intuitively be broken down as an alternation of two phases. The first one tries to lift the measures of positions outside the quasi dominion $\mathsf{Q}(\varrho)$ in order to extend it, while the second one lifts the positions inside $\mathsf{Q}(\varrho)$ that can be forced to exit from it by player 1. The algorithm terminates when no new position can be absorbed within the quasi dominion and no measure needs to be lifted to allow the 1-winning positions to exit from it, when possible. To this end, we define a controlled lift operator $\mathsf{lift}\colon \mathrm{QDR} \times 2^{\mathrm{Ps}} \times 2^{\mathrm{Ps}} \rightharpoonup \mathrm{QDR}$ that works on QDRs and takes two additional parameters, a source and a target set of positions. The intended meaning is that we want to restrict the application of the lift operation to the positions in the source set S, while using only the moves leading to the target set T. The different nature of the two types of lifting operations is reflected in the actual values of the source and target parameters.

$$\mathsf{lift}(\varrho, \mathrm{S}, \mathrm{T}) \triangleq \varrho^{\star}, \text{ where}$$

$$\mu_{\varrho^{\star}}(v) \triangleq \begin{cases} \mathsf{max}\{\mu_{\varrho}(u) + v : u \in Mv(v) \cap \mathrm{T}\}, & \text{if } v \in \mathrm{S} \cap \mathrm{Ps}_{\mathrm{o}}; \\ \mathsf{min}\{\mu_{\varrho}(u) + v : u \in Mv(v) \cap \mathrm{T}\}, & \text{if } v \in \mathrm{S} \cap \mathrm{Ps}_{1}; \\ \mu_{\varrho}(v), & \text{otherwise;} \end{cases}$$

and, for all 0-positions $v \in \mathsf{Q}(\varrho^{\star}) \cap \mathrm{Ps}_{\mathrm{o}}$,

$$\sigma_{\varrho^{\star}}(v) \in \underset{u \in Mv(v) \cap \mathrm{T}}{\mathsf{argmax}} \ \mu_{\varrho}(u) + v, \text{ if } \mu_{\varrho^{\star}}(v) \neq \mu_{\varrho}(v), \text{ and } \sigma_{\varrho^{\star}}(v) = \sigma_{\varrho}(v), \text{ otherwise.}$$

Except for the restriction on the outgoing moves considered, which are those leading to the targets in T, the lift operator acts on the measure component of a QDR very much like the original lift operator does. In order to ensure that the result is still a QDR, however, the lift operator must also update the 0-witness of the quasi dominion. This is required to guarantee that Conditions 1a and 1c of Definition 9.3.2 are preserved. If the measure of a 0-position v is not affected by the lift, the 0-witness must not change for that position. On the other hand, if the application of the lift operation increases the measure, then the 0-witness on v needs to be updated to any move (v, u) that grants measure $\mu_{\varrho^{\star}}(v)$ to v. In principle, more than one such move may exist and any one of them can serve the purpose as witness.

The solution algorithm can then be expressed as the inflationary fixpoint [Bou49, Wit50] of the composition of the two phases mentioned above, defined by the progress operators $\mathsf{prg}_{\mathrm{o}}$ and prg_{+}.

$$\mathsf{sol} \triangleq \mathsf{ifp}\, \varrho \,.\, \mathsf{prg}_{+}(\mathsf{prg}_{\mathrm{o}}(\varrho))\colon \mathrm{QDR} \rightharpoonup \mathrm{QDR}.$$

The first phase is computed by the operator $\mathsf{prg}_{\mathrm{o}} \colon \mathrm{QDR} \rightharpoonup \mathrm{QDR}$, defined as follows:

$$\mathsf{prg}_{\mathrm{o}}(\varrho) \triangleq \mathsf{sup}\{\varrho, \mathsf{lift}(\varrho, \overline{\mathsf{Q}(\varrho)}, \mathrm{Ps})\}.$$

This operator is responsible of enforcing the progress condition on the positions outside the quasi dominion $\mathsf{Q}(\varrho)$ that do not satisfy the inequalities between the measures along a move leading to $\mathsf{Q}(\varrho)$ itself. It does that by applying the lift operator with $\overline{\mathsf{Q}(\varrho)}$ as source and no restrictions on the moves. Those position that acquire a positive measure in this phase contribute to enlarging the current quasi dominion. Observe that the strategy component of the QDRis updated so that it is a 0-witness of the new quasi dominion. To guarantee that measures never decrease, the supremum *w.r.t.* the QDR-space ordering is taken as result.

Lemma 9.3.1. *Let $\varrho \in \mathrm{QDR}$ be a fixpoint of $\mathsf{prg}_{\mathrm{o}}$. Then, μ_ϱ is a progress measure over $\overline{\mathsf{Q}(\varrho)}$.*

Proof. By definition of the progress operator $\mathsf{prg}_{\mathrm{o}}$, we have that $\varrho = \mathsf{prg}_{\mathrm{o}}(\varrho) = \mathsf{sup}\{\varrho, \mathsf{lift}(\varrho, \overline{\mathsf{Q}(\varrho)}, \mathrm{Ps})\}$, from which we derive $\varrho^\star \triangleq \mathsf{lift}(\varrho, \overline{\mathsf{Q}(\varrho)}, \mathrm{Ps}) \sqsubseteq \varrho$. Now, consider an arbitrary position $v \in \overline{\mathsf{Q}(\varrho)}$ and observe that $\mu_{\varrho^\star}(v) \leq \mu_\varrho(v)$, due to Item 2 of Definition 9.3.2. At this point, the proof proceeds by a case analysis on the owner of the position v itself.

- $[v \in \mathrm{Ps}_{\mathrm{o}}]$**.** By definition of the lift operator, we have that $\mu_\varrho(u) + v \leq \mathsf{max}\{\mu_\varrho(u) + v : u \in Mv(v)\} = \mu_{\varrho^\star}(v)$, for all adjacents $u \in Mv(v)$ of v. Thus, $\mu_\varrho(u) + v \leq \mu_{\varrho^\star}(v) \leq \mu_\varrho(v)$, thanks to the above observation. Consequently, Condition 1 of Definition 9.2.2 is satisfied on $\overline{\mathsf{Q}(\varrho)}$.
- $[v \in \mathrm{Ps}_1]$**.** Again by definition of the lift operator, we have that $\mu_\varrho(u) + v \leq \mathsf{min}\{\mu_\varrho(u) + v : u \in Mv(v)\} = \mu_{\varrho^\star}(v)$, for some adjacent $u \in Mv(v)$ of v. Due to the above observation, it holds that $\mu_\varrho(u) + v \leq \mu_{\varrho^\star}(v) \leq \mu_\varrho(v)$. Hence, Condition 2 of Definition 9.2.2 is satisfied on $\overline{\mathsf{Q}(\varrho)}$ as well.

□

The second phase, instead, implements the mechanism intuitively described above, while analyzing the simple example of Figure 9.1. This is achieved by the operator prg_{+} reported in Algorithm 21. The procedure iteratively examines the current quasi dominion by lifting the measures of the positions that must exit from it. Specifically, it processes $\mathsf{Q}(\varrho)$ layer by layer, starting from the outer layer of positions that must escape from. The process ends when a, possibly empty, closed weak quasi dominion is obtained. Recall that all the positions in a closed weak quasi dominion are necessarily winning for player 0, due to Proposition 9.3.1. We distinguish two sets of positions in $\mathsf{Q}(\varrho)$. Those that already satisfy the progress condition and those that do not. The measures of first ones already witness an escape route

from $\mathsf{Q}(\varrho)$. The other ones, instead, are those whose current choice is to remain inside it. For instance, when considering the measure function μ_2 in the example of Figure 9.1, position $\mathtt{a}$ belongs to the first set, while positions $\mathtt{c}$ and $\mathtt{d}$ to the second one, since the choice of $\mathtt{c}$ is to follow the internal move $(\mathtt{c}, \mathtt{d})$.

Since the only positions that change measure are those in the second set, only such positions need to be examined. To identify them, which form a weak quasi dominion $\Delta(\varrho)$ strictly contained in $\mathsf{Q}(\varrho)$, we proceed as follows. First, we collect the set $\mathsf{npp}(\varrho)$ of positions in $\mathsf{Q}(\varrho)$ that do not satisfy the progress condition, called the *non-progress positions*. Then, we compute the set of positions that will have no choice other than reaching $\mathsf{npp}(\varrho)$. The non-progress positions are computed as follows.

$$\begin{aligned}\mathsf{npp}(\varrho) \triangleq \{v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_0 : \exists u \in Mv(v) \,.\, \mu_\varrho(v) < \mu_\varrho(u) + v\} \\ \cup \{v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_1 : \forall u \in Mv(v) \,.\, \mu_\varrho(v) < \mu_\varrho(u) + v\}.\end{aligned}$$

The remaining positions in $\Delta(\varrho)$ are collected as the inflationary fixpoint of the following operator.

$$\begin{aligned}\mathsf{pre}(\varrho, \mathrm{Q}) \triangleq \mathrm{Q} \cup \{v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_0 : \sigma_\varrho(v) \in \mathrm{Q}\} \\ \cup \{v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_1 : \forall u \in Mv(v) \setminus \mathrm{Q} \,.\, \mu_\varrho(v) < \mu_\varrho(u) + v\}.\end{aligned}$$

The final result is

$$\Delta(\varrho) \triangleq (\mathsf{ifp}\, \mathrm{Q} \,.\, \mathsf{pre}(\varrho, \mathrm{Q}))(\mathsf{npp}(\varrho))$$

Intuitively, $\Delta(\varrho)$ contains all the 0-positions that are forced to reach $\mathsf{npp}(\varrho)$ via the quasi-dominion 0-witness and all the 1-positions that can only avoid reaching $\mathsf{npp}(\varrho)$ by strictly increasing their measure, which player 1 wants obviously to prevent.

It is important to observe that, from a functional view-point, the progress operator prg_+ would work just as well if applied to the entire quasi dominion $\mathsf{Q}(\varrho)$, since it would simply leave unchanged the measure of those positions that already satisfy the progress condition. However, it is crucial that only the positions in $\Delta(\varrho)$ are processed in order to achieve the best asymptotic complexity bound known to date. We shall reiterate on this point later on.

Algorithm 21: Progress Operator

```
   signature prg+ : QDR ⇀ QDR
   function prg+(ϱ)
1      Q ← Δ(ϱ)
2      while esc(ϱ, Q) ≠ ∅ do
3          E ← bep(ϱ, Q)
4          ϱ ← lift(ϱ, E, Q̄)
5          Q ← Q \ E
6      ϱ ← win(ϱ, Q)
7      return ϱ
```

At each iteration of the while-loop of Algorithm 21, let Q denote the current (weak) quasi dominion, initially set to $\Delta(\varrho)$ (Line 1). It first identifies the positions in Q that can immediately escape from it (Line 2).

Those are *(i)* all the 1-position with a move leading outside of Q and *(ii)* the 0-positions v whose 0-witness σ_ϱ forces v to exit from Q, namely $\sigma_\varrho(v) \notin \mathrm{Q}$, and that cannot strictly increase their measure by choosing to remain in Q. While the condition for 1-position is obvious, the one for 0-positions require some explanation. The crucial observation here is that, while player 0 does indeed prefer to remain in the quasi dominion, it can only do so while ensuring that by changing strategy it does not enable infinite plays within Q that are winning for the adversary. In other words, the new 0-strategy must still be a 0-witness for Q and this can only be ensured if the new choice strictly increases its measure. The operator $\mathsf{esc}\colon \mathrm{QDR} \times 2^{\mathrm{Ps}} \to 2^{\mathrm{Ps}}$ formalizes the idea:

$$\begin{aligned}\mathsf{esc}(\varrho, \mathrm{Q}) \triangleq\ & \{v \in \mathrm{Q} \cap \mathrm{Ps}_1 : Mv(v) \setminus \mathrm{Q} \neq \emptyset\} \\ & \cup \{v \in \mathrm{Q} \cap \mathrm{Ps}_0 : \sigma_\varrho(v) \notin \mathrm{Q} \wedge \forall u \in Mv(v) \cap \mathrm{Q} \,.\, \mu_\varrho(u) + v \leq \mu_\varrho(v))\}.\end{aligned}$$

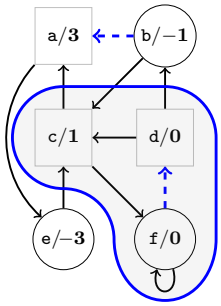


Figure 9.2: Another MPG.

Consider, for instance, the example in Figure 9.2 and a QDRϱ such that $\mu_\varrho = \{\mathtt{a} \mapsto 3; \mathtt{b} \mapsto 2; \mathtt{c}, \mathtt{d}, \mathtt{f} \mapsto 1; \mathtt{e} \mapsto 0\}$ and $\sigma_\varrho = \{\mathtt{b} \mapsto \mathtt{a}; \mathtt{f} \mapsto \mathtt{d}\}$. In this case, we have $\mathrm{Q}_\varrho = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{f}\}$ and $\Delta(\varrho) = \{\mathtt{c}, \mathtt{d}, \mathtt{f}\}$, since `c` is the only non-progress positions, `d` is forced to follow `c` in order to avoid the measure increase required to reach `b`, and `f` is forced by the 0-witness to reach `d`. Now, consider the situation where the current weak quasi dominion is $\mathrm{Q} = \{\mathtt{c}, \mathtt{f}\}$, *i.e.* after `d` has escaped from $\Delta(\varrho)$. The escape set of Q is $\{\mathtt{c}, \mathtt{f}\}$. To see why the 0-position `f` is escaping, observe that $\mu_\varrho(\mathtt{f}) + \mathtt{f} = 1 = \mu_\varrho(\mathtt{f})$ and that, indeed, should player 0 choose to change its strategy and take the move $(\mathtt{f}, \mathtt{f})$ to remain in Q, it would obtain an infinite play with payoff 0, thus violating the definition of weak quasi dominion.

Before proceeding, we want to stress an easy consequence of the definition of the notion of escape set and Conditions 1c and 1d of Definition 9.3.2, *i.e.*, that every escape position of the quasi dominion $\mathsf{Q}(\varrho)$ can only assume its weight as possible measure inside a QDRϱ, as reported is the following proposition. This observation, together with Proposition 9.3.2, precisely ensures that the measure of a position $v \in \mathsf{Q}(\varrho)$ is an under approximation of the weight of all finite plays leaving $\mathsf{Q}(\varrho)$.

Proposition 9.3.4. *Let ϱ be a* QDR*. Then, $\mu_\varrho(v) = \mathsf{wg}(v) > 0$, for all $v \in \mathsf{esc}(\varrho, \mathsf{Q}(\varrho))$.*

Now, going back to the analysis of the algorithm, if the escape set is non-empty, we need to select the escape positions that need to be lifted in order to satisfy the progress condition. The main difficulty is to do so in such a way that the resulting measure function still satisfies Condition 1d of Definition 9.3.2, for all the 1-positions with positive measure. The problem occurs when a 1-position can exit either

immediately or passing through a path leading to another position in the escape set. Consider again the example above, where $\mathrm{Q} = \Delta(\varrho) = \{\mathtt{c}, \mathtt{d}, \mathtt{f}\}$. If position $\mathtt{d}$ immediately escapes from Q using the move $(\mathtt{d}, \mathtt{b})$, it would change its measure to $\mu'(\mathtt{d}) = \mu(\mathtt{b}) + \mathtt{d} = 2 > \mu(\mathtt{d}) = 1$. Now, position $\mathtt{c}$ has two ways to escape, either directly with move $(\mathtt{c}, \mathtt{a})$ or by reaching the other escape position $\mathtt{d}$ passing through $\mathtt{f}$. The first choice would set its measure to $\mu(\mathtt{a}) + \mathtt{c} = 4$. The resulting measure function, however, would not satisfy Condition 1d of Definition 9.3.2, as the new measure of $\mathtt{c}$ would be greater than $\mu'(\mathtt{d}) + \mathtt{c} = 2$, preventing to obtain a QDR. Similarly, if position $\mathtt{d}$ escapes from Q passing through $\mathtt{c}$ via the move $(\mathtt{c}, \mathtt{a})$, we would have $\mu''(\mathtt{d}) = \mu''(\mathtt{c}) + \mathtt{d} = (\mu(\mathtt{a}) + \mathtt{c}) + \mathtt{d} = 4 > 2 = \mu(\mathtt{b}) + \mathtt{d}$, still violating Condition 1d. Therefore, in this specific case, the only possible way to escape is to reach $\mathtt{b}$. The solution to this problem is simply to lift in the current iteration only those positions that obtain the lowest possible measure increase, hence position $\mathtt{d}$ in the example, leaving the lift of $\mathtt{c}$ to some subsequent iteration of the algorithm that would choose the correct escape route via $\mathtt{d}$. To do so, we first compute the minimal measure increase, called the *best-escape forfeit*, that each position in the escape set would obtain by exiting the quasi dominion immediately. The positions with the lowest possible forfeit, called *best-escape positions*, can all be lifted at the same time. The intuition is that the measure of all the positions that escape from a (weak) quasi dominion will necessarily be increased of at least the minimal best-escape forfeit. This observation is at the core of the proof of Theorem 9.3.1 (see the appendix) ensuring that the desired properties of QDRs are preserved by the operator prg_+. The set of best-escape positions is computed by the operator $\mathsf{bep}\colon \mathrm{QDR} \times 2^{\mathrm{Ps}} \to 2^{\mathrm{Ps}}$ as follows:

$$\mathsf{bep}(\varrho, \mathrm{Q}) \triangleq \operatorname*{\mathsf{argmin}}_{v \in \mathsf{esc}(\varrho, \mathrm{Q})} \mathsf{bef}(\mu_\varrho, \mathrm{Q}, v),$$

where the operator $\mathsf{bef}\colon \mathrm{MF} \times 2^{\mathrm{Ps}} \times \mathrm{Ps} \to \mathbb{N}_\infty$ computes, for each position v in a quasi dominion Q, its best-escape forfeit:

$$\mathsf{bef}(\mu, \mathrm{Q}, v) \triangleq \begin{cases} \mathsf{max}\{\mu(u) + v - \mu(v) : u \in Mv(v) \setminus \mathrm{Q}\}, & \text{if } v \in \mathrm{Ps}_\mathrm{o}; \\ \mathsf{min}\{\mu(u) + v - \mu(v) : u \in Mv(v) \setminus \mathrm{Q}\}, & \text{otherwise.} \end{cases}$$

In our example, $\mathsf{bef}(\mu, \mathrm{Q}, \mathtt{c}) = \mu(\mathtt{a}) + \mathtt{c} - \mu(\mathtt{c}) = 4 - 1 = 3$, while $\mathsf{bef}(\mu, \mathrm{Q}, \mathtt{d}) = \mu(\mathtt{b}) + \mathtt{d} - \mu(\mathtt{d}) = 2 - 1 = 1$. Therefore, $\mathsf{bep}(\varrho, \mathrm{Q}) = \{\mathtt{d}\}$.

Once the set E of best-escape positions is identified (Line 3 of the algorithm), the procedure simply lifts them restricting the possible moves to those leading outside the current quasi dominion (Line 4). Those positions are, then, removed from the set (Line 5), thus obtaining a smaller weak quasi dominion ready for the next iteration.

The algorithm terminates when the (possibly empty) current quasi dominion Q is closed. By virtue of Proposition 9.3.1, all those positions belong to Wn_o and their measure is set to ∞ by means of the operator $\mathsf{win}\colon \mathrm{QDR} \times 2^{\mathrm{Ps}} \rightharpoonup \mathrm{QDR}$ (Line 6), which also computes the winning 0-strategy on those positions.

$$\mathsf{win}(\varrho, \mathrm{Q}) \triangleq \varrho^\star, \text{ where}$$

$$\mu_{\varrho^\star} \triangleq \mu_\varrho[\mathrm{Q} \mapsto \infty]$$

and, for all 0-positions $v \in \mathsf{Q}(\varrho^\star) \cap \mathrm{Ps}_\mathrm{o}$,

$$\sigma_{\varrho^\star}(v) \in \operatorname*{\mathsf{argmax}}_{u \in Mv(v) \cap \mathrm{Q}} \mu_\varrho(u) + v, \text{ if } \sigma_\varrho(v) \notin \mathrm{Q} \text{ and } \sigma_{\varrho^\star}(v) = \sigma_\varrho(v), \text{ otherwise.}$$

Observe that, since we know that every 0-position $v \in \mathrm{Q} \cap \mathrm{Ps}_\mathrm{o}$, whose current 0-witness leads outside Q, is not an escape position, any move (v, u) within Q that grants the maximal stretch $\mu_\varrho(u) + v$ strictly increases its measure and, therefore, is a possible choice for a 0-witness of the 0-dominion Q.

At this point, it should be quite evident that the progress operator prg_+ is responsible of enforcing the progress condition on the positions inside the quasi dominion $\mathsf{Q}(\varrho)$, thus, the following necessarily holds.

Lemma 9.3.2. *Let $\varrho \in \mathrm{QDR}$ be a fixpoint of prg_+. Then, μ_ϱ is a progress measure over $\mathsf{Q}(\varrho)$.*

Proof. Let us consider the infinite monotone sequence of position sets $\mathrm{Q}_\mathrm{o} \supseteq \mathrm{Q}_1 \supseteq \ldots$ defined as follows: $\mathrm{Q}_\mathrm{o} \triangleq \mathsf{\Delta}(\varrho)$; $\mathrm{Q}_{i+1} \triangleq \mathrm{Q}_i \setminus \mathrm{E}_i$, where $\mathrm{E}_i \triangleq \mathsf{bep}(\varrho, \mathrm{Q}_i)$, for all $i \in \mathbb{N}$. Since $|\mathrm{Q}_\mathrm{o}| < \infty$, there necessarily exists an index $k \in \mathbb{N}$ such that $\mathrm{Q}_{k+1} = \mathrm{Q}_k$. By definition of the progress operator prg_+ and the equality $\varrho = \mathsf{prg}_+(\varrho)$, we have that $\varrho = \mathsf{lift}(\varrho, \mathrm{E}_i, \overline{\mathrm{Q}_i})$, for all $i \in [0, k[$, and $\varrho = \mathsf{win}(\varrho, \mathrm{Q}_k)$. Now, consider an arbitrary position $v \in \mathsf{Q}(\varrho)$. If $v \notin \mathsf{\Delta}(\varrho)$, due to the definition of the set $\mathsf{\Delta}(\varrho)$, the position v satisfies by definition of the appropriate condition of Definition 9.2.2 on $\mathsf{Q}(\varrho)$. Therefore, let us assume $v \in \mathsf{\Delta}(\varrho)$. Then, it is obvious that either $v \in \mathrm{Q}_k$ or there is a unique index $i \in [0, k[$ such that $v \in \mathrm{Q}_i \setminus \mathrm{Q}_{i+1}$, *i.e.*, $v \in \mathrm{E}_i$. In the first case, we have $\mu_\varrho(v) = \infty$, due to the definition of the function win. Therefore, v is a progress position. In the other case, the proof proceeds by a case analysis on the owner of the position v itself.

- $[v \in \mathrm{Ps}_\mathrm{o}]$**.** First observe that $\mathsf{bep}(\varrho, \mathrm{Q}_i) \subseteq \mathsf{esc}(\varrho, \mathrm{Q}_i)$. Thus, due to the definition of the function esc, we have that $\mu_\varrho(u) + v \leq \mu_\varrho(v)$, for all positions $u \in Mv(v) \cap \mathrm{Q}_i$. Now, by the definition of the lift operator, we have that $\mu_\varrho(u) + v \leq \mathsf{max}\{\mu_\varrho(u) + v : u \in Mv(v) \cap \overline{\mathrm{Q}_i}\} = \mu_\varrho(v)$, for all adjacents $u \in Mv(v) \cap \overline{\mathrm{Q}_i}$ of v. Consequently, $\mu_\varrho(u) + v \leq \mu_\varrho(v)$, for all positions $u \in Mv(v)$, as required by Condition 1 of Definition 9.2.2 on $\mathsf{Q}(\varrho)$.

- $[v \in \mathrm{Ps}_1]$. Again by definition of the lift operator, we have that $\mu_\varrho(u) + v \leq \mathsf{min}\{\mu_\varrho(u) + v : u \in Mv(v) \cap \overline{\mathsf{Q}_i}\} = \mu_\varrho(v)$, for some adjacent $u \in Mv(v) \cap \overline{\mathsf{Q}_i} \subseteq Mv(v)$ of v. Hence, Condition 2 of Definition 9.2.2 is satisfied on $\mathsf{Q}(\varrho)$ as well.

□

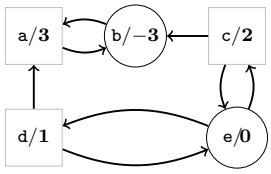


Figure 9.3: Yet another MPG.

We now exemplify the lack of monotonicity of the progress operator prg_+. To do so, consider the game of Figure 9.3 and the following two QDRs ϱ_1 and ϱ_2 defined via their components: $\mu_{\varrho_1} = \{\mathsf{a} \mapsto 3; \mathsf{b} \mapsto 0; \mathsf{c} \mapsto 2; \mathsf{d}, \mathsf{e} \mapsto 1\}$ and $\sigma_{\varrho_1} = \{\mathsf{e} \mapsto \mathsf{d}\}$; $\mu_{\varrho_2} = \{\mathsf{a} \mapsto 3; \mathsf{b} \mapsto 0; \mathsf{c}, \mathsf{e} \mapsto 2; \mathsf{d} \mapsto 1\}$ and $\sigma_{\varrho_2} = \{\mathsf{e} \mapsto \mathsf{c}\}$. Obviously, $\varrho_1 \sqsubset \varrho_2$. However, $\varrho_1^\star \triangleq \mathsf{prg}_+(\varrho_1) \not\sqsubseteq \varrho_2^\star \triangleq \mathsf{prg}_+(\varrho_2)$. Indeed, $\mu_{\varrho_1^\star} = \{\mathsf{a} \mapsto 3; \mathsf{b} \mapsto 0; \mathsf{c} \mapsto 2; \mathsf{d}, \mathsf{e} \mapsto 4\}$, while $\mu_{\varrho_2^\star} = \{\mathsf{a} \mapsto 3; \mathsf{b} \mapsto 0; \mathsf{c}, \mathsf{e} \mapsto 2; \mathsf{d}, \mapsto 3\}$, which implies that $\varrho_2^\star \sqsubset \varrho_1^\star$. Moreover, $\varrho_1^\star$ is already a progress measure, while $\varrho_2^\star$ requires another application of prg_+ in order to solve the game, since $\varrho_1^\star = \mathsf{prg}_+(\varrho_2^\star)$.

In order to prove the correctness of the proposed algorithm, we first need to ensure that any quasi-dominion space $\mathcal{Q}$ is indeed closed under the operators prg_o and prg_+.

In the following, for a finite path π of an MPG, we denote by $\mathsf{fst}(\pi)$ and $\mathsf{lst}(\pi)$ the first and last positions of π, respectively.

Lemma 9.3.3. *Let $\varrho \in$ QDR and $\sigma^\star \in \mathrm{Str}_\mathrm{o}(\mathsf{Q}(\varrho))$ a 0-strategy such that, if $\sigma^\star(v) \neq \sigma_\varrho(v)$, then $\mu_\varrho(v) < \mu_\varrho(\sigma^\star(v)) + v$, for all positions $v \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_\mathrm{o}$. Then, $\sigma^\star$ is a 0-witness for $\mathsf{Q}(\varrho)$.*

Proof. The proof proceed by induction on the number $i \triangleq |\mathrm{D}|$ of the positions in $\mathrm{D} \triangleq \{v \in \mathrm{Ps}_\mathrm{o} : \sigma^\star(v) \neq \sigma_\varrho(v)\}$ on which the two strategies $\sigma^\star$ and σ_ϱ differ. The base case $i = 0$ is immediate, since ϱ is a QDR. Therefore, assume $i > 0$, let $v \in \mathrm{D}$, and consider the strategy $\widehat{\sigma} \in \mathrm{Str}_\mathrm{o}(\mathsf{Q}(\varrho))$ such that $\widehat{\sigma}(v) = \sigma_\varrho(v)$ and $\widehat{\sigma}(u) = \sigma^\star(u)$, for all positions $u \in \mathsf{Q}(\varrho) \cap \mathrm{Ps}_\mathrm{o}$ with $u \neq v$. By the inductive hypothesis, we have that $\widehat{\sigma}$ is a 0-witness for the quasi dominion $\mathsf{Q}(\varrho)$. Now, consider an arbitrary path π compatible with the 0-strategy $\sigma^\star$. If π does not meet v, it is necessarily compatible with the 0-strategy $\widehat{\sigma}$, thus, $\mathsf{wg}(\pi) > 0$. If π meets v once, then it can be decomposed as $\pi' v \pi''$, where π' and π'' are paths not meeting v, where only the first can be possibly empty. On the one hand, if π'' is infinite, by Proposition 9.3.1, we have $\mathsf{wg}(\pi'') = \infty$ and, so, $\mathsf{wg}(\pi) = \mathsf{wg}(\pi' v \pi'') = \infty$. On the other hand, if π'' is finite, then, by Propositions 9.3.2 and 9.3.4, we have that $0 < \mu_\varrho(\mathsf{fst}(\pi' v)) \leq \mathsf{wg}(\pi') + \mu_\varrho(\mathsf{lst}(\pi' v)) = \mathsf{wg}(\pi') + \mu_\varrho(v)$ and $\mu_\varrho(\mathsf{fst}(\pi'')) \leq \mathsf{wg}(\pi'')$, since both π' and π'' are compatible with $\widehat{\sigma}$. Moreover, $\mu_\varrho(v) < \mu_\varrho(\sigma^\star(v)) + v = \mu_\varrho(\mathsf{fst}(\pi'')) + v = \mu_\varrho(\mathsf{fst}(\pi'')) + \mathsf{wg}(v)$. Now, by putting all things together, we

have $0 < \mu_\varrho(\mathsf{fst}(\pi' v)) \leq \mathsf{wg}(\pi') + \mu_\varrho(v) < \mathsf{wg}(\pi') + \mu_\varrho(\mathsf{fst}(\pi'')) + \mathsf{wg}(v) \leq \mathsf{wg}(\pi') + \mathsf{wg}(v) + \mathsf{wg}(\pi'') = \mathsf{wg}(\pi' v \pi'')$, *i.e.*, $\mathsf{wg}(\pi) > 0$. Finally, consider the case where π meets v more than once and, so, infinitely many times, due to the regularity of the path, which is in its turn due to the memoryless strategies. Then, π can be written as $\pi'(v\pi'')^\omega = \pi' v(\pi'' v)^\omega$, where π' and π'' are possibly empty paths not meeting v. First observe that the finite path $\pi'' v$ is compatible with $\widehat{\sigma}$, thus, by Proposition 9.3.2, we have that $\mu_\varrho(\mathsf{fst}(\pi'' v)) \leq \mathsf{wg}(\pi'') + \mu_\varrho(v)$. Moreover, $\mu_\varrho(v) < \mu_\varrho(\mathsf{fst}(\pi'' v)) + \mathsf{wg}(v)$, as already shown above. Hence, $\mu_\varrho(v) < \mu_\varrho(\mathsf{fst}(\pi'')) + \mathsf{wg}(v) \leq \mathsf{wg}(\pi'') + \mathsf{wg}(v) + \mu_\varrho(v) = \mathsf{wg}(\pi'' v) + \mu_\varrho(v)$, which implies $\mathsf{wg}(\pi'' v) > 0$. As a consequence, $\mathsf{wg}((\pi'' v)^\omega) = \infty$ and, so, $\mathsf{wg}(\pi) = \mathsf{wg}(\pi' v(\pi'' v)^\omega) > 0$. Summing up, $\sigma^\star$ is a 0-witness for the quasi dominion $\mathsf{Q}(\varrho)$ as required by the lemma statement. □

The following theorem states that the operators are total functions on $\mathcal{Q}$.

Theorem 9.3.1 (Totality)**.** *The progress operators* $\mathsf{prg}_{\mathrm{o}}$ *and* prg_{+} *are total inflationary functions.*

Proof. The proof proceeds by showing that, for each $\varrho \in \mathrm{QDR}$, the elements $\mathsf{prg}_{\mathrm{o}}(\varrho)$ and $\mathsf{prg}_{+}(\varrho)$ are QDRtoo. We also prove that $\varrho \sqsubseteq \mathsf{prg}_{\mathrm{o}}(\varrho)$ and $\varrho \sqsubseteq \mathsf{prg}_{+}(\varrho)$. The two operators are analyzed separately.

- $[\mathsf{prg}_{\mathrm{o}}]$**.** Let $\varrho^\star \triangleq \mathsf{prg}_{\mathrm{o}}(\varrho) = \mathsf{sup}\{\varrho, \mathsf{lift}(\varrho, \overline{\mathsf{Q}(\varrho)}, \mathrm{Ps})\} \sqsupseteq \varrho$. It is obvious, so, that $\mathsf{prg}_{\mathrm{o}}$ is inflationary. Consider now a position $v \in \mathsf{Q}(\varrho^\star)$. Recall that $\mu_{\varrho^\star}(v) > 0$. If $v \in \mathsf{Q}(\varrho)$, by definition of the lift operator, it holds that $\mu_{\varrho^\star}(v) = \mu_\varrho(v)$ and $\sigma_{\varrho^\star}(v) = \sigma_\varrho(v)$, thus the appropriate condition between Conditions 1c and 1d of Definition 9.3.2 is verified, since $\varrho \in \mathrm{QDR}$. Thus, assume $v \in \overline{\mathsf{Q}(\varrho)}$. If $v \in \mathrm{Ps}_{\mathrm{o}}$, we have that $\mu_{\varrho^\star}(v) = \mathsf{max}\{\mu_\varrho(u) + v : u \in Mv(v)\} = \mu_\varrho(\sigma_{\varrho^\star}(v)) + v = \mu_{\varrho^\star}(\sigma_{\varrho^\star}(v)) + v$, since $\sigma_{\varrho^\star}(v) \in \mathsf{Q}(\varrho)$. As a consequence, Condition 1c is satisfied. If $v \in \mathrm{Ps}_1$, instead, we have that $\mu_{\varrho^\star}(v) = \mathsf{min}\{\mu_\varrho(u) + v : u \in Mv(v)\}$, which implies $\mu_{\varrho^\star}(v) \leq \mu_\varrho(u) + v = \mu_{\varrho^\star}(u) + v$, for all adjacents $u \in Mv(v)$, as required by Condition 1d. To complete the proof that $\mathsf{prg}_{\mathrm{o}}$ is a total function from QDR to itself, we need to show that $\varrho^\star$ satisfies Conditions 1b and 1a too. It is immediate to see that $\|\mu_\varrho\|_{\mathrm{o}} \subseteq \|\mu_{\varrho^\star}\|_{\mathrm{o}}$. Since ϱ is a QDR, $\|\mu_\varrho\|_{\mathrm{o}}$ is a 0-dominion. Moreover, for all positions $v \in \|\mu_{\varrho^\star}\|_{\mathrm{o}} \setminus \|\mu_\varrho\|_{\mathrm{o}}$, it holds that $\sigma_{\varrho^\star}(v) \in \|\mu_\varrho\|_{\mathrm{o}}$, if $v \in \mathrm{Ps}_{\mathrm{o}}$, and $Mv(v) \subseteq \|\mu_\varrho\|_{\mathrm{o}}$, otherwise. Therefore, $\|\mu_{\varrho^\star}\|_{\mathrm{o}}$ is necessarily a 0-dominion, so Condition 1b is verified. Finally, let us focus on Condition 1a and consider a $(\sigma_{\varrho^\star}, v)$-play $v\pi$. If, on the one hand, π is infinite and does not meet v, thanks to Proposition 9.3.1, we have $\mathsf{wg}(\pi) = \infty$, thus $\mathsf{wg}(v\pi) = \infty$ and, so, $\mathsf{wg}(v\pi) > 0$. If π is finite, instead, it holds that $\mathsf{lst}(\pi) \in \mathsf{esc}(\varrho, \mathsf{Q}(\varrho))$ and, so, $\mu_{\varrho^\star}(\mathsf{lst}(\pi)) = \mathsf{wg}(\mathsf{lst}(\pi))$, due to Proposition 9.3.4. Now, by Proposition 9.3.2, we have that $\mu_\varrho(\mathsf{fst}(\pi)) \leq \mu_\varrho(\mathsf{lst}(\pi)) + \mathsf{wg}(\pi_{<\ell-1}) = \mathsf{wg}(\mathsf{lst}(\pi)) + \mathsf{wg}(\pi_{<\ell-1}) = \mathsf{wg}(\pi)$, where $\ell \in \mathbb{N}$ is the length of π Moreover, $0 < \mu_\varrho(v) \leq \mu_\varrho(\mathsf{fst}(\pi)) + v = \mu_\varrho(\mathsf{fst}(\pi)) + \mathsf{wg}(v)$, thanks to the previously proved

Conditions 1c and 1d. Hence, $0 < \mu_\varrho(v) \leq \mu_\varrho(\mathsf{fst}(\pi)) + \mathsf{wg}(v) \leq \mathsf{wg}(v) + \mathsf{wg}(\pi) = \mathsf{wg}(v\pi)$, as required by the definition of quasi 0-dominion. Finally, if π is infinite and does meet v, it can be decomposed as $(v\pi')^\omega$, where π is a non-empty finite path that does not meet v. Then, by exploiting the same reasoning done above for the case where π is finite, we have that $\mathsf{wg}(v\pi') > 0$, which implies $\mathsf{wg}(\pi) = \mathsf{wg}((v\pi')^\omega) = \infty$.

- $[\mathsf{prg}_+]$**.** Let $\varrho^\star \triangleq \mathsf{prg}_+(\varrho)$ and consider the two infinite monotone sequences $\mathrm{Q}_0 \supseteq \mathrm{Q}_1 \supseteq \ldots$ and $\varrho_0 \sqsubseteq \varrho_1 \sqsubseteq \ldots$ defined as follows: $\mathrm{Q}_0 \triangleq \Delta(\varrho)$ and $\varrho_0 \triangleq \varrho$; $\mathrm{Q}_{i+1} \triangleq \mathrm{Q}_i \setminus \mathrm{E}_i$ and $\varrho_{i+1} = \mathsf{lift}(\varrho_i, \mathrm{E}_i, \overline{\mathrm{Q}_i})$, where $\mathrm{E}_i \triangleq \mathsf{bep}(\varrho_i, \mathrm{Q}_i) \subseteq \mathsf{esc}(\varrho_i, \mathrm{Q}_i)$, for all $i \in \mathbb{N}$. Since $|\mathrm{Q}_0| < \infty$, there necessarily exists an index $k \in \mathbb{N}$ such that $\mathrm{Q}_{k+1} = \mathrm{Q}_k$, $\varrho_{k+1} = \varrho_k$. Moreover, observe that $\varrho^\star = \mathsf{win}(\varrho_k, \mathrm{Q}_k)$. We first prove, by induction on the index $i \in \mathbb{N}$ of the sequences, that every ϱ_i satisfies Conditions 1a and 1c of Definition 9.3.2. Finally, we show that $\varrho^\star$ is a QDR.

  The base case $i = 0$ is trivial, since $\varrho_i = \varrho$ is a QDR. Now, let us consider the inductive case $i > 0$. Since the lift operator only modifies the measure of positions belonging to $\mathrm{E}_{i-1} \subseteq \mathrm{Q}_{i-1} \subseteq \Delta(\varrho) \subseteq \mathsf{Q}(\varrho)$, it immediately follows that $\mathsf{Q}(\varrho_i) = \mathsf{Q}(\varrho_{i-1}) = \mathsf{Q}(\varrho)$. Moreover, if $\sigma_{\varrho_i}(v) \neq \sigma_{\varrho_{i-1}}(v)$, we have that $\mu_{\varrho_{i-1}}(v) < \mu_{\varrho_i}(v) = \mu_{\varrho_i}(\sigma_{\varrho_i}(v)) = \mu_{\varrho_{i-1}}(\sigma_{\varrho_i}(v))$, for all positions $v \in \mathsf{Q}(\varrho_i) \cap \mathrm{Ps}_0$, where the latter equality is due to the fact that $\sigma_{\varrho_i}(v) \notin \mathrm{E}_{i-1}$. Thus, by Lemma 9.3.3, it holds that σ_{ϱ_i} is a 0-witness for $\mathsf{Q}(\varrho_i)$, *i.e.*, Condition 1a is verified. Also, Condition 1c directly follows from the definition of the 0-strategy inside the lift operator.

  At this point, we can conclude the proof by showing that $\varrho^\star$ is a QDR. Indeed, by Lemma 9.3.3, $\sigma_{\varrho^\star}$ is a 0-witness for $\mathsf{Q}(\varrho_k) = \mathsf{Q}(\varrho)$, so, Condition 1a is satisfied. Similarly to the inductive analysis developed above, Condition 1c directly follows from the definition of the 0-strategy inside the win function. Moreover, the set Q_k is a closed subset of $\mathsf{Q}(\varrho_k)$, since $\mathrm{E}_k = \emptyset$ and, so, $\mathsf{esc}(\varrho_k, \mathrm{Q}_k) = \emptyset$. Therefore, $\mathrm{Q}_k \subseteq \mathrm{Wn}_0$, by Proposition 9.3.1. In addition, all positions in $\|\mu_{\varrho_k}\|_0 \setminus (\|\mu\|_0 \cup \mathrm{Q}_k)$ necessarily reach $(\|\mu\|_0 \cup \mathrm{Q}_k) \subseteq \mathrm{Wn}_0$. As a consequence, Condition 1b is verified as well.

  It remains to prove Condition 1d. To do so, let $f_i \triangleq \mathsf{min}_{v \in \mathsf{esc}(\varrho_i, \mathrm{Q}_i)} \mathsf{bef}(\mu_{\varrho_i}, \mathrm{Q}_i, v)$. We now first show that the sequence of natural numbers $f_0, f_1, \ldots$ is monotone, *i.e.*, $f_i \leq f_{i+1}$. Suppose by contradiction that $f_i > f_{i+1}$, for some index $i \in \mathbb{N}$. Then, there necessarily exists a position $v \in \mathsf{esc}(\varrho_{i+1}, \mathrm{Q}_{i+1}) \setminus \mathsf{esc}(\varrho_i, \mathrm{Q}_i)$ with $v \in \mathrm{E}_{i+1}$ such that $f_{i+1} = \mathsf{bef}(\mu_{\varrho_{i+1}}, \mathrm{Q}_{i+1}, v) < f_i$. We proceeds by a case analysis on the owner of the position v.

  - $[v \in \mathrm{Ps}_0]$**.** By definition of the best-escape forfeit function, we have that $f_{i+1} = \mathsf{max}\{\mu_{\varrho_{i+1}}(u) + v - \mu_{\varrho_{i+1}}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \geq \mu_{\varrho_{i+1}}(\sigma_{\varrho_i}(v)) + v - \mu_{\varrho_{i+1}}(v)$, since $\sigma_{\varrho_i}(v) \in \mathrm{E}_i$ and, so,

$\sigma_{\varrho_i}(v) \notin \mathrm{Q}_{i+1}$. Therefore, the following equalities and inequalities hold, which lead to the contradiction $f_i \leq f_{i+1} < f_i$:

$$\begin{aligned} f_{i+1} &\geq \mu_{\varrho_{i+1}}(\sigma_{\varrho_i}(v)) + v - \mu_{\varrho_{i+1}}(v) \\ &= \mu_{\varrho_{i+1}}(\sigma_{\varrho_i}(v)) + \mathsf{wg}(v) - \mu_{\varrho_{i+1}}(v) \\ &= \mu_{\varrho_i}(\sigma_{\varrho_i}(v)) + f_i + \mathsf{wg}(v) - \mu_{\varrho_{i+1}}(v) \\ &= \mu_{\varrho_i}(\sigma_{\varrho_i}(v)) + f_i + \mathsf{wg}(v) - \mu_{\varrho_i}(v) \\ &= \mu_{\varrho_i}(\sigma_{\varrho_i}(v)) + v - \mu_{\varrho_i}(v) + f_i \\ &\geq f_i. \end{aligned}$$

Notice that the first and last equality are due to the definition of the measure stretch operator. The second one is derived from the fact that $\sigma_{\varrho_i}(v) \in \mathrm{E}_i$, while the third one from $v \in \mathrm{E}_{i+1}$, which implies $\mu_{\varrho_{i+1}}(v) = \mu_{\varrho_i}(v)$. Finally, the last inequality follows from Condition 1c applied to ϱ_i, *i.e.*, $\mu_{\varrho_i}(v) \leq \mu_{\varrho_i}(\sigma_{\varrho_i}(v)) + v$.

– $[v \in \mathrm{Ps}_1]$**.** Again by definition of the best-escape forfeit function, we have that $f_{i+1} = \mathsf{min}\{\mu_{\varrho_{i+1}}(u) + v - \mu_{\varrho_{i+1}}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\}$. In addition, $Mv(v) \setminus \mathrm{Q}_{i+1} \subseteq \mathrm{E}_i$ Therefore, the following equalities hold:

$$\begin{aligned} f_{i+1} &= \mathsf{min}\{\mu_{\varrho_{i+1}}(u) + v - \mu_{\varrho_{i+1}}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \\ &= \mathsf{min}\{\mu_{\varrho_{i+1}}(u) + \mathsf{wg}(v) - \mu_{\varrho_{i+1}}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \\ &= \mathsf{min}\{\mu_{\varrho_i}(u) + f_i + \mathsf{wg}(v) - \mu_{\varrho_{i+1}}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \\ &= \mathsf{min}\{\mu_{\varrho_i}(u) + f_i + \mathsf{wg}(v) - \mu_{\varrho_i}(v) : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \\ &= \mathsf{min}\{\mu_{\varrho_i}(u) + v - \mu_{\varrho_i}(v) + f_i : u \in Mv(v) \setminus \mathrm{Q}_{i+1}\} \\ &\geq f_i. \end{aligned}$$

Notice that the second and last equality are due to the definition of the measure stretch operator. The third one is derived from the fact that $u \in Mv(v) \setminus \mathrm{Q}_{i+1} \subseteq \mathrm{E}_i$, while the fourth one from $v \in \mathrm{E}_{i+1}$, which implies $\mu_{\varrho_{i+1}}(v) = \mu_{\varrho_i}(v)$. Finally, the last inequality follows from Condition 1d applied to ϱ, *i.e.*, $\mu_{\varrho_i}(v) = \mu_{\varrho}(v) \leq \mu_{\varrho}(u) + v \leq \mu_{\varrho_i}(u) + v$, for all adjacents $u \in Mv(v)$.

Now suppose by contradiction that Condition 1d does not hold for $\varrho^\star$. Then, there exist a 1-position $v \in \mathsf{Q}(\varrho^\star) \cap \mathrm{Ps}_1$ and one of its adjacents $u \in Mv(v)$ such that $\mu_{\varrho^\star}(u) + v < \mu_{\varrho^\star}(v)$. Due to the

process used to compute $\varrho^\star$, there are indexes $i, j \in [0, k]$ such that $\mu_{\varrho^\star}(u) = \mu_{\varrho_{i+1}}(u) = \mu_\varrho(u) + f_i$ and $\mu_{\varrho^\star}(v) = \mu_{\varrho_{j+1}}(v) = \mu_\varrho(v) + f_j$. Now, by Condition 1d applied to ϱ, we have $\mu_\varrho(v) \leq \mu_\varrho(u) + v$, which implies that $0 \leq \mu_\varrho(u) + v - \mu_\varrho(v) < f_j - f_i$ and, consequently, both $i < j$ and $u \notin \mathrm{Q}_j$. However,

$$\begin{aligned} f_j - f_i &= \mathsf{min}\{\mu_{\varrho_j}(z) + v - \mu_{\varrho_j}(v) : z \in Mv(v) \setminus \mathrm{Q}_j\} - f_i \\ &\leq \mu_{\varrho_j}(u) + v - \mu_{\varrho_j}(v) - f_i \\ &= \mu_{\varrho_j}(u) + v - \mu_\varrho(v) - f_i \\ &= \mu_{\varrho_{i+1}}(u) + v - \mu_\varrho(v) - f_i \\ &= (\mu_\varrho(u) + f_i) + v - \mu_\varrho(v) - f_i \\ &= \mu_\varrho(u) + v - \mu_\varrho(v), \end{aligned}$$

leading to the contradiction $\mu_\varrho(u) + v - \mu_\varrho(v) < f_j - f_i \leq \mu_\varrho(u) + v - \mu_\varrho(v)$. Notice that the first equality is due to the definition of the best-escape forfeit function. The second and third ones, instead, follows from the fact that v and u changed their values at iterations $j + 1$ and $i + 1$, respectively. Finally, the fourth equality derives from the operation of lift and best-escape forfeit computed on u.

□

Since both operators are inflationary, so is their composition, which admits fixpoint. Therefore, the operator sol is well defined. Moreover, following the same considerations discussed at the end of Section 9.2, it can be proved the fixpoint is obtained after at most $n \cdot (S + 1)$ iterations. We first prove the following necessary lemma.

Lemma 9.3.4. *Let $\varrho^\star \triangleq \mathsf{prg}_+(\varrho)$, for some $\varrho \in \mathrm{QDR}$, and $S \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathrm{Ps} \wedge \mathsf{wg}(v) > 0\}$. Then, for all positions $v \in \mathsf{Q}(\varrho^\star)$ with $\varrho^\star(v) \neq \infty$, it holds that $\varrho^\star(v) \leq S$.*

Proof. Suppose by contradiction that there exists a position $v \in \mathsf{Q}(\varrho^\star)$ with $\varrho^\star(v) \neq \infty$, but $\varrho^\star(v) > S$. If $v \in \mathsf{Q}(\varrho) \setminus \mathsf{\Delta}(\varrho)$, then $\varrho^\star(v) = \varrho(v)$. Moreover, there exists a finite path π compatible with the 0-strategy σ_ϱ and entirely contained in $\mathsf{Q}(\varrho) \setminus \mathsf{\Delta}(\varrho)$, which starts in v and ends in $\mathsf{esc}(\varrho, \mathsf{Q}(\varrho))$, *i.e.*, $\mathsf{fst}(\pi) = v$ and $\mathsf{lst}(\pi) \in \mathsf{esc}(\varrho, \mathsf{Q}(\varrho))$. By Propositions 9.3.2 and 9.3.4, we have that $S < \mu_\varrho(v) \leq \mathsf{wg}(\pi) \leq S^\star$, with $S^\star \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathsf{Q}(\varrho) \setminus \mathsf{\Delta}(\varrho) \wedge \mathsf{wg}(v) > 0\}$, where the last inequality is obviously due to the fact that there are no repeated positions in π, being it finite. However, $S^\star \leq S$, which means that a contradiction has been reached with $v \in \mathsf{Q}(\varrho) \setminus \mathsf{\Delta}(\varrho)$. Thus, assume $v \in \mathsf{\Delta}(\varrho)$ and

consider the two infinite monotone sequences $\mathrm{Q}_0 \supseteq \mathrm{Q}_1 \supseteq \ldots$ and $\varrho_0 \sqsubseteq \varrho_1 \sqsubseteq \ldots$ defined as in the proof of Theorem 9.3.1: $\mathrm{Q}_0 \triangleq \Delta(\varrho)$ and $\varrho_0 \triangleq \varrho$; $\mathrm{Q}_{i+1} \triangleq \mathrm{Q}_i \setminus \mathrm{E}_i$ and $\varrho_{i+1} = \mathsf{lift}(\varrho_i, \mathrm{E}_i, \overline{\mathrm{Q}_i})$, where $\mathrm{E}_i \triangleq \mathsf{bep}(\varrho_i, \mathrm{Q}_i) \subseteq \mathsf{esc}(\varrho_i, \mathrm{Q}_i)$, for all $i \in \mathbb{N}$. Also, let $S_0 \leq S_1 \leq \ldots < \top$ be the sequence of natural numbers defined as $S_i \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathsf{Q}(\varrho) \setminus \mathrm{Q}_i \wedge \mathsf{wg}(v) > 0\}$. Since $\varrho^\star(v) \neq \infty$, there exists an index k such that $v \in \mathrm{E}_k$ with $\mathsf{bef}(\varrho_k, \mathrm{Q}_k, v) < \infty$. Therefore, to prove the thesis, it suffices to show that $\mu_{\varrho_i}(z) \leq S_i$, for all positions $z \in \mathrm{Q}_i$ and index $i \in [0, k]$. The base case $i = 0$ follows by applying the same reasoning previously done for the case $v \in \mathsf{Q}(\varrho) \setminus \Delta(\varrho)$ and by noticing that $S_0 = S^\star$. Now, let $i > 0$. By definition of the lift operator, there exists at least one adjacent x of z such that $\mu_{\varrho_{i+1}}(z) = \mu_{\varrho_i}(x) + z$. By the inductive hypothesis, $\mu_{\varrho_i}(x) \leq S_i$. Thus, $\mu_{\varrho_{i+1}}(z) \leq S_i + \mathsf{wg}(z) \leq S_{i+1}$, since $\mathsf{wg}(z) \notin S_i$. □

Let $\mathsf{ifp}_k\, X \,.\, \mathsf{F}(X)$ denote the k-th iteration of an inflationary operator F. Then, we have the following theorem.

Theorem 9.3.2 (Termination)**.** *The solver operator* $\mathsf{sol} \triangleq \mathsf{ifp}\, \varrho \,.\, \mathsf{prg}_+(\mathsf{prg}_0(\varrho))$ *is a well-defined total function. Moreover, for every* $\varrho \in \mathrm{QDR}$ *it holds that* $\mathsf{sol}(\varrho) = (\mathsf{ifp}_k\, \varrho^\star \,.\, \mathsf{prg}_+(\mathsf{prg}_0(\varrho^\star)))(\varrho)$*, for some index* $k \leq n{\cdot}(S{+}1)$*, where* n *is the number of positions in the* MPG *and* $S \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathrm{Ps} \wedge \mathsf{wg}(v) > 0\}$ *the total sum of its positive weights.*

Proof. Consider the sequence $\varrho_0, \varrho_1, \ldots$ recursively defined as follows: $\varrho_0 \triangleq (\mathsf{ifp}_0\, \varrho^\star \,.\, \mathsf{prg}_+(\mathsf{prg}_0(\varrho^\star)))(\varrho) = \varrho$ and $\varrho_{i+1} \triangleq (\mathsf{ifp}_{i+1}\, \varrho^\star \,.\, \mathsf{prg}_+(\mathsf{prg}_0(\varrho^\star)))(\varrho) = \mathsf{prg}_+(\mathsf{prg}_0(\varrho_i))$, for all $i \in \mathbb{N}$. By induction on the index i, thanks to the totality and inflationary properties of the progress operators prg_0 and prg_+ previously proved in Theorem 9.3.1, one can easily show that every ϱ_i is a QDRsatisfying $\varrho_i \sqsubseteq \varrho_{i+1}$. Moreover, by Lemma 9.3.4, we have that $\varrho_i(v) \leq S$, for all positions $v \in \mathsf{Q}(\varrho_i)$ with $\varrho_i(v) \neq \infty$ and index $i > 0$. Now, there are at most $n \cdot (S + 1)$ such QDRs, thus, there necessarily exists an index $k \leq n \cdot (S + 1)$ such that $\varrho_{k+1} = \varrho_k$, which implies $\mathsf{sol}(\varrho) = (\mathsf{ifp}\, \varrho^\star \,.\, \mathsf{lift}(\varrho^\star))(\varrho) = \varrho_k$. Hence, the thesis immediately follows. □

Consider, as a final example, the game depicted in Figure 9.4, with $k > 2$, where the numbers denote the weights of the positions of the game, in the picture labeled (0), and the measures assigned by the procedure, in the remaining ones. Each picture also features both the 0-witness strategy in dashed blue and the best counter 1-strategy in dashed red for the current quasi dominion. Moreover, solid colored moves are moves along which the measure strictly increases. Below each picture, we also indicate the phase, prg_0 or prg_+, that produces the displayed result. The computation starts from the initial $\mathrm{QDR}\varrho_0 = (\mu_0, \sigma_0)$, assigning measure 0 to all the positions of the game with the associated empty strategy. The first iteration applies prg_0 to ϱ_0, which lifts positions `a`, `f`, and `g` to their respective weights, leading to ϱ_1 as shown in Picture (1). At this point, $\mathsf{Q}(\varrho_1) = \{\mathtt{a}, \mathtt{f}, \mathtt{g}\}$ but $\Delta(\varrho_1)$ is empty, as all those positions

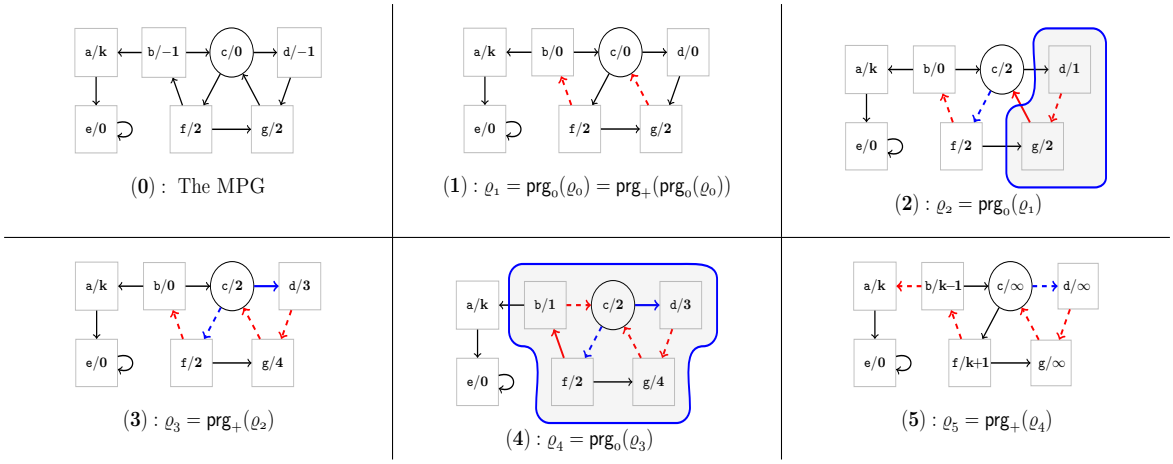


Figure 9.4: A simulation.

already satisfy the progress condition, thus, prg_+ does nothing. In the next iteration, prg_0 applied to ϱ_1 results in the lifting of positions $\mathtt{c}$ and $\mathtt{d}$, as reported in Picture (2). Position $\mathtt{c}$ is a 0-position and the lift operator chooses $(\mathtt{c}, \mathtt{f})$ as its strategy. The resulting quasi-dominion is $\mathsf{Q}(\varrho_2) = \{\mathtt{a}, \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}\}$ and $\Delta(\varrho_2) = \{\mathtt{d}, \mathtt{g}\}$, with $\mathtt{g}$ the only escape position that is also non-progress. The measure of $\mathtt{g}$ is lifted to $\mu_2(\mathtt{c}) + \mathtt{g} = 4$. Finally, it is the turn of position $\mathtt{d}$ to be lifted to $\mu_2(\mathtt{g}) + \mathtt{d} = 3$. Picture (3) shows the resulting $\textsc{qdr}\varrho_3$. The final iteration first applies prg_0 to ϱ_3 (Picture (4)), lifting position $\mathtt{b}$ to measure 1 via the move $(\mathtt{b}, \mathtt{c})$. This change of measure triggers another application of prg_+, as position $\mathtt{f}$ is now non-progress. The resulting $\textsc{qdr}\varrho_4$ is such that $\mathsf{Q}(\varrho_4) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}\}$ and $\Delta(\varrho_4) = \{\mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}\}$. The only escape position is $\mathtt{b}$, which is lifted directly to measure $k - 1$. In the remaining set $\{\mathtt{c}, \mathtt{d}, \mathtt{f}, \mathtt{g}\}$, the only escape position is $\mathtt{f}$, which is lifted to measure $k + 1$. The resulting weak quasi dominion $\{\mathtt{c}, \mathtt{d}, \mathtt{g}\}$, however, is closed, since $\mu_{\varrho_4}(\mathtt{c}) = 2 < \mu_{\varrho_4}(\mathtt{d}) + \mathtt{c} = 3$. Therefore, player 0 changes strategy and chooses the move $(\mathtt{c}, \mathtt{d})$. Since no escape positions remain, the set $\{\mathtt{c}, \mathtt{d}, \mathtt{g}\}$ is winning for player 0 and the win operator lifts all their measures to ∞, leading to ϱ_5 in Picture (5). The measure function μ_5 is now a progress measure and the algorithm terminates. The total number of single measure updates for QDPM to reach the fixpoint on the example of Figure 9.4 is 13, regardless of the value of the maximal weight k in the game assigned to position $\mathtt{a}$.

On the other hand, it can easily be proved that SEPM [BCD+11] requires $3k + 8$ applications of its lift operator to compute a progress measure, for a total of $5k + 9$ measure updates. Indeed, the first two evaluations of lift, starting from μ_0, lead to $\mu_2 = \{\mathtt{a} \mapsto k; \mathtt{b}, \mathtt{e} \mapsto 0; \mathtt{c}, \mathtt{f}, \mathtt{g} \mapsto 2; \mathtt{d} \mapsto 1\}$, as in Picture (2), and require 5 measure lifts. Then, the algorithm iteratively increases the measures of $\mathtt{b}$, $\mathtt{g}$, $\mathtt{d}$, $\mathtt{f}$, and $\mathtt{c}$ by

applying $3(k-1)$ times the lift operator, for a total of $5(k-1)$ measure lifts: $\mu_{3i} = \mu_{3i-1}[\mathtt{b} \mapsto i; \mathtt{g} \mapsto i+3]$, $\mu_{3i+1} = \mu_{3i}[\mathtt{d}, \mathtt{f} \mapsto i+2]$, and $\mu_{3i+2} = \mu_{3i+1}[\mathtt{c} \mapsto i+2]$, for all $i \in [1, k-1]$. At this point, $\mathtt{b}$ and $\mathtt{f}$ have obtained measures $k-1$ and $k+1$, respectively, which suffice to satisfy the progress relation along the moves $(\mathtt{f}, \mathtt{b})$ and $(\mathtt{b}, \mathtt{a})$. However, the 1-position $\mathtt{g}$ does not satisfy such a relation along its unique move $(\mathtt{g}, \mathtt{c})$, since $\mu_{3k-1}(\mathtt{g}) = k+2 < \mu_{3k-1}(\mathtt{c}) + \mathtt{g} = (k+1)+2 = k+3$. Therefore, other six applications of lift are needed before $\mathtt{g}$ can exceed the bound $S = \mathsf{wg}(\mathtt{a}) + \mathsf{wg}(\mathtt{f}) + \mathsf{wg}(\mathtt{g}) = k+4$. Each one of them modifies the measure of one position only, for a total of 6 lifts: $\mu_{3(k+i)} = \mu_{3(k+i)-1}[\mathtt{g} \mapsto k+3+i]$, $\mu_{3(k+i)+1} = \mu_{3(k+i)}[\mathtt{d} \mapsto k+2+i]$, and $\mu_{3(k+i)+2} = \mu_{3(k+i)+1}[\mathtt{c} \mapsto k+2+i]$, for $i \in \{0, 1\}$. At this point, we have $\mu_{3k+6} = \mu_{3k+5}[\mathtt{g} \mapsto \infty]$, $\mu_{3k+7} = \mu_{3k+6}[\mathtt{d} \mapsto \infty]$, and, finally, $\mu_{3k+8} = \mu_{3k+7}[\mathtt{c} \mapsto \infty]$, which contribute with the remaining 3 lifts. From this observation, the next result immediately follows.

Theorem 9.3.3 (Efficiency)**.** *An infinite family of* MPG*s* $\{\mathfrak{I}_k\}_k$ *exists on which* QDPM *requires a constant number of measure updates, while* SEPM*requires* $\mathrm{O}(k)$ *such updates.*

From Theorem 9.2.1, together with Lemmas 9.3.1 and 9.3.2, it follows that the solution provided by the algorithm is indeed a progress measure, hence establishing soundness.

Lemma 9.3.5. *Let* $\varrho^\star \triangleq \mathsf{sol}(\varrho)$ *be the result of the solver operator applied to an arbitrary* $\varrho \in \mathrm{QDR}$*. Then,* $\varrho^\star$ *is a fixpoint of the progress operators,* i.e.*,* $\varrho^\star = \mathsf{prg}_{\mathrm{o}}(\varrho^\star) = \mathsf{prg}_{+}(\varrho^\star)$*.*

Proof. By definition of inflationary fixpoint, $\varrho^\star$ is a fixpoint of the composition of the two progress operators, *i.e.*, $\varrho^\star = \mathsf{prg}_{+}(\mathsf{prg}_{\mathrm{o}}(\varrho^\star))$, which are inflationary functions, due to Theorem 9.3.1. As a consequence, we have that $\varrho^\star = \mathsf{prg}_{+}(\mathsf{prg}_{\mathrm{o}}(\varrho^\star)) \sqsupseteq \mathsf{prg}_{\mathrm{o}}(\varrho^\star) \sqsupseteq \varrho^\star$. Thus, $\mathsf{prg}_{\mathrm{o}}(\varrho^\star) = \varrho^\star$ and, so, $\mathsf{prg}_{+}(\varrho^\star) = \varrho^\star$. □

Theorem 9.3.4 (Soundness)**.** $\|\mathsf{sol}(\varrho)\|_1 \subseteq \mathrm{Wn}_1$*, for every* $\varrho \in \mathrm{QDR}$*.*

Proof. Let $\varrho^\star \triangleq \mathsf{sol}(\varrho)$ be the result of the solver operator applied to $\varrho \in \mathrm{QDR}$. By Lemma 9.3.5, it holds that $\varrho^\star = \mathsf{prg}_{\mathrm{o}}(\varrho^\star) = \mathsf{prg}_{+}(\varrho^\star)$. As a consequence, $\mu_{\varrho^\star}$ is a progress measure, due to Lemmas 9.3.1 and 9.3.2. At this point, by recalling that $\|\varrho^\star\|_1 = \|\mu_{\varrho^\star}\|_1$, as reported in Definition 9.3.2, the thesis is immediately derived by applying Theorem 9.2.1 to $\mu_{\varrho^\star}$. □

On the other hand, Theorem 9.3.2, together with Condition 1b of Definition 9.3.2, ensures that all the positions with infinite measure are winning for player 0, hence the algorithm is also complete.

Theorem 9.3.5 (Completeness)**.** $\|\mathsf{sol}(\varrho)\|_{\mathrm{o}} \subseteq \mathrm{Wn}_{\mathrm{o}}$*, for every* $\varrho \in \mathrm{QDR}$*.*

Proof. The thesis immediately follows by considering Theorem 9.3.2 and Condition 1b of Definition 9.3.2. Indeed, by the statement of the recalled theorem, $\mathsf{sol}(\varrho)$ is a QDR, independently of the element $\varrho \in \text{QDR}$ given as input to the solver operator. Thus, thanks to the above condition, it holds the $\|\mathsf{sol}(\varrho)\|_{\text{o}} \subseteq \text{Wn}_{\text{o}}$. □

The following lemma ensures that each execution of the operator prg_{+} strictly increases the measure of all the positions in $\Delta(\varrho)$.

Lemma 9.3.6. *Let $\varrho^{\star} \triangleq \mathsf{prg}_{+}(\varrho)$, for some $\varrho \in \text{QDR}$. Then, $\mu_{\varrho^{\star}}(v) > \mu_{\varrho}(v)$, for all positions $v \in \Delta(\varrho)$.*

Proof. Consider the set $\text{E} \triangleq \mathsf{bep}(\varrho, \Delta(\varrho)) \subseteq \mathsf{esc}(\varrho, \Delta(\varrho))$ and let $\widehat{\varrho} \triangleq \mathsf{lift}(\varrho, \text{E}, \overline{\Delta(\varrho)})$. First observe that $\mu_{\widehat{\varrho}}(v) = \mu_{\varrho^{\star}}(v)$, for all escape positions $v \in \text{E}$. We now show that $\mu_{\varrho^{\star}}(v) > \mu_{\varrho}(v)$, via a case analysis on the owner of the position v itself.

- $[v \in \text{Ps}_{\text{o}}]$**.** By definition of the function esc, it holds that $\sigma_{\varrho}(v) \notin \Delta(\varrho)$ and $\mu_{\varrho}(v) \geq \mu_{\varrho}(u) + v$, for all adjacents $u \in Mv(v) \cap \Delta(\varrho)$. Since $v \in \Delta(\varrho)$, due to the way this specific weak quasi dominion is constructed, $v \in \mathsf{npp}(\varrho)$. Thus, there exists a successor $u^{\star} \in Mv(v)$ with $\mu_{\varrho}(v) < \mu_{\varrho}(u^{\star}) + v$, from which it follows that $u^{\star} \notin \Delta(\varrho)$, *i.e.*, $u^{\star} \in \overline{\Delta(\varrho)}$. As a consequence, we obtain that $\mu_{\widehat{\varrho}}(v) \geq \mu_{\varrho}(u^{\star}) + v > \mu_{\varrho}(v)$. Hence, $\mu_{\varrho^{\star}}(v) > \mu_{\varrho}(v)$.
- $[v \in \text{Ps}_{1}]$**.** Since $v \in \Delta(\varrho)$, we have that $\mu_{\varrho}(v) < \mu_{\varrho}(u) + v$, for all adjacents $u \in Mv(v) \setminus \Delta(\varrho)$. Thus, $\mu_{\widehat{\varrho}}(v) = \mathsf{min}\{\mu_{\varrho}(u) + v : u \in Mv(v) \setminus \Delta(\varrho)\} > \mu_{\varrho}(v)$. Hence, $\mu_{\varrho^{\star}}(v) > \mu_{\varrho}(v)$ in this case as well.

Now, consider a position $v \in \Delta(\varrho) \setminus \text{E}$. Obviously, $\mu_{\varrho}(v) < \infty$. If $\mu_{\varrho^{\star}}(v) = \infty$, the thesis immediately follows. Otherwise, it will be considered as an escape of some weak quasi dominion $\text{Q} \subset \Delta(\varrho)$, after the removal of the first escape positions in E. Due to the non-decreasing property of the sequence of best-escape forfeit shown in the proof of Theorem 9.3.1, v exits from Q with a forfeit $f^{\star}$ at least as great as the one f of E that we just proved to be strictly positive. Indeed, $f = \mu_{\varrho^{\star}}(z) - \mu_{\varrho}(z) > 0$, for all $z \in \text{E}$. Therefore, $\mu_{\varrho^{\star}}(v) - \mu_{\varrho}(v) = f^{\star} \geq f > 0$, which implies $\mu_{\varrho^{\star}}(v) > \mu_{\varrho}(v)$. □

Recall that each position can at most be lifted $S+1 = \text{O}(n \cdot W)$ times and, by the previous lemma, the complexity of sol only depends on the cumulative cost of such lift operations. We can express, then, the total cost as the sum, over the set of positions in the game, of the cost of all the lift operations performed on that positions. Each such operation can be computed in time linear in the number of incoming and outgoing moves of the corresponding lifted position v, namely $\text{O}\big((|Mv(v)| + |Mv^{-1}(v)|) \cdot \log S\big)$,

with $\mathrm{O}(\log S)$ the cost of each arithmetic operation involved. Summing all up, the actual asymptotic complexity of the procedure can, therefore, be expressed as $\mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W))$.

Theorem 9.3.6 (Complexity)**.** QDPM *requires time* $\mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W))$ *to solve an* MPG *with* n *positions,* m *moves, and maximal positive weight* W*.*

Proof. To compute sol efficiently, we now provide an imperative reformulation of the functional fixpoint algorithm $\mathsf{sol} \triangleq \mathsf{ifp}\, \varrho \,.\, \mathsf{prg}_+(\mathsf{prg}_{\mathrm{o}}(\varrho))$ with the desired complexity. Recall that, by Lemma 9.3.4, each position can only be lifted $S + 1$ times, where $S \triangleq \sum\{\mathsf{wg}(v) \in \mathbb{N} : v \in \mathrm{Ps} \wedge \mathsf{wg}(v) > 0\} = \mathrm{O}(n \cdot W)$. Therefore, to obtain the claimed complexity, we have to guarantee that the cost of all the computational steps be linear in the number of measure increases. To do so, it suffices to ensure that the algorithm explores the incoming and outgoing moves only of those positions whose measures are actually lifted. This is clearly the case for the lift operator itself, since it only explores the outgoing moves of each position in its source set. The only remaining problem is to be able to identify the positions that need to be lifted in the next iteration, by only exploring the incoming moves of the positions just lifted. Solving this problem requires some technical tricks. Specifically, inspired by [BCD+11], will employ vectors of counters, namely c, d and g, that associates with 0-positions the number of moves that do not satisfy the progress condition, and with 1-positions the number of moves that satisfy it. In addition, we will also use a priority queue T to allow an efficient identification of the *best-escape positions* during the computation of the operator prg_+.

Algorithm 22: MPGSolver

signature $\mathsf{sol}\colon \mathrm{MPG} \to \mathrm{QDR}$

procedure $\mathsf{sol}(\supset)$

1 $\quad \varrho \leftarrow (\{v \in \mathrm{Ps} \mapsto 0\}, \varnothing)$

2 $\quad \mathsf{c} \leftarrow \{v \in \mathrm{Ps}_{\boxminus} \mapsto |\{u \in Mv(v) : \mu_\varrho(v) \geq \mu_\varrho(u) + v\}|\}$

3 $\quad (\mathrm{N_o}, \mathrm{N_+}) \leftarrow (\{v \in \mathrm{Ps} : \mathsf{wg}(v) > 0\}, \emptyset)$

4 $\quad$ **while** $\mathrm{N_o} \neq \emptyset \vee \mathrm{N_+} \neq \emptyset$ **do**

5 $\quad\quad (\mathrm{N_o}, \mathrm{A}) \leftarrow \mathsf{prg}_{\mathrm{o}}(\mathrm{N_o})$

6 $\quad\quad \mathrm{N_+} \leftarrow \mathrm{N_+} \cup \mathrm{A}$

7 $\quad\quad (\mathrm{A}, \mathrm{N_+}) \leftarrow \mathsf{prg}_+(\mathrm{N_+})$

8 $\quad\quad \mathrm{N_o} \leftarrow \mathrm{N_o} \cup \mathrm{A}$

9 $\quad$ **return** ϱ

Algorithm 22 reports the procedural implementation of $\mathsf{sol}(\varrho_{\mathrm{o}})$, where ϱ_{o} is the smallest possible QDR,

as defined at Line 1. At the beginning of each iteration $i \in \mathbb{N}$ of the while-loop at Line 4, the variable ϱ maintains the $\textsc{qdr}\varrho_i$ computed by applying to ϱ_{o} the composition $\mathsf{prg}_+ \circ \mathsf{prg}_{\mathrm{o}}$ i times. Moreover, the sets N_{o} and N_+ contain, respectively, the positions that need to be lifted by $\mathsf{prg}_{\mathrm{o}}$ and the non-progress positions in ϱ_i. The formal invariants at Line 4 are: $\mathrm{N}_{\mathrm{o}} = \{v \in \mathrm{Ps} : \mu_{\varrho_i}(v) = 0 \neq \mu_{\varrho_{i+1}}(v)\}$ and $\mathrm{N}_+ = \mathsf{npp}(\varrho_i)$. Observe that these invariants are trivially satisfied for $i = 0$, thanks to Line 3. After the execution of the progress procedure $\mathsf{prg}_{\mathrm{o}}$ at Line 5, whose code is reported in Algorithm 23, we have that $\mathrm{N}_{\mathrm{o}} \subseteq \{v \in \mathrm{Ps} : \mu_{\varrho_{i+1}}(v) = 0 \neq \mu_{\varrho_{i+2}}(v)\}$ and $\mathrm{N}_+ \cup \mathrm{A} = \mathsf{npp}(\varrho_i^\star)$, where $\varrho_i^\star \triangleq \mathsf{prg}_{\mathrm{o}}(\varrho_i)$. Thus, Line 6 ensures that $\mathrm{N}_+ = \mathsf{npp}(\varrho_i^\star)$. Line 7 calls the progress procedure prg_+, which is reported in Algorithm 25, and forces the lift of the measures of all the positions in $\Delta(\varrho_i^\star)$, as stated by Lemma 9.3.6. In addition, the verified invariants are $\mathrm{N}_{\mathrm{o}} \cup \mathrm{A} = \{v \in \mathrm{Ps} : \mu_{\varrho_{i+1}}(v) = 0 \neq \mu_{\varrho_{i+2}}(v)\}$ and $\mathrm{N}_+ = \mathsf{npp}(\varrho_{i+1})$. Finally, after Line 8, it holds that $\mathrm{N}_{\mathrm{o}} = \{v \in \mathrm{Ps} : \mu_{\varrho_{i+1}}(v) = 0 \neq \mu_{\varrho_{i+2}}(v)\}$, as required by the previously discussed invariants for the next iteration $i + 1$. Observe that Line 2 is used to initialize, for each 1-position $v \in \mathrm{Ps}_1$, the counter $\mathsf{c}(v)$ to the number of adjacents $u \in Mv(v)$ of v that satisfy the progress inequality $\mu_{\varrho_{\mathrm{o}}}(v) \geq \mu_{\varrho_{\mathrm{o}}}(u) + v$.

The subsequent analysis of Algorithms 23 and 25 shows that the procedures $\mathsf{prg}_{\mathrm{o}}(\varrho, \mathsf{c}, \mathrm{N}_{\mathrm{o}})$ and $\mathsf{prg}_+(\varrho, \mathsf{c}, \mathrm{N}_+)$ require time

$$\mathrm{O}\left(\sum_{v \in \mathrm{N}_{\mathrm{o}}} (|Mv(v)| + |Mv^{-1}(v)|) \cdot \log S\right) \text{ and } \mathrm{O}\left(\sum_{v \in \Delta(\varrho)} (|Mv(v)| + |Mv^{-1}(v)|) \cdot \log S\right),$$

respectively, where $\mathsf{npp}(\varrho) = \mathrm{N}_+$. In particular, the factor $\log S$ is due to all the arithmetic operations required to compute the stretch of the measures. Since during the entire execution of the algorithm each position $v \in \mathrm{Ps}$ can appear at most once in some N_{o} and at most S times in some $\Delta(\varrho)$, it follows that the total cost of Algorithm 22 is

$$\mathrm{O}\left(n + (S+1) \cdot \sum_{v \in \mathrm{Ps}} (|Mv(v)| + |Mv^{-1}(v)|) \cdot \log S\right) = \mathrm{O}(n + S \cdot m \cdot \log S) = \mathrm{O}(n \cdot m \cdot W \cdot \log(n \cdot W)),$$

where the term n is due to the initialization operations at Lines 1-3.

Observe that, the two procedures $\mathsf{prg}_{\mathrm{o}}$ and prg_+, together with the auxiliary one reported in Algorithm 24, share with Algorithm 22 both the current $\textsc{qdr}\varrho$ and the counter c as global variables.

Algorithm 23: Efficient Progress Zero Operator

signature $\mathsf{prg}_{\mathrm{o}} \colon 2^{\mathrm{Ps}} \to 2^{\mathrm{Ps}} \times 2^{\mathrm{Ps}}$

procedure $\mathsf{prg}_{\mathrm{o}}(\mathrm{N})$

1 $\quad \mathrm{Z} \leftarrow \emptyset$

2 $\quad \widehat{\mu} \leftarrow \{v \in \mathrm{N} \mapsto \mu_{\varrho}(v)\}$

3 $\quad \varrho \leftarrow \mathsf{sup}\{\varrho, \mathsf{lift}(\varrho, \mathrm{N}, \mathrm{Ps})\}$

4 $\quad \mathsf{c} \leftarrow \mathsf{c}[v \in \mathrm{N} \cap \mathrm{Ps}_{\boxminus} \mapsto |\{u \in Mv(v) : \mu_{\varrho}(v) \geq \mu_{\varrho}(u) + v\}|]$

5 $\quad$ **foreach** $(v, u) \in Mv;\ u \in \mathrm{N};\ \mu_{\varrho}(v) < \mu_{\varrho}(u) + v$ **do**

6 $\quad\quad$ **if** $v \in \mathrm{Ps}_{\oplus}$ **then**

7 $\quad\quad\quad \mathrm{Z} \leftarrow \mathrm{Z} \cup v$

$\quad\quad$ **else**

8 $\quad\quad\quad$ **if** $v \notin \mathrm{N} \wedge \mu_{\varrho}(v) \geq \widehat{\mu}(u) + v$ **then** $\mathsf{c}(v) \leftarrow \mathsf{c}(v) - 1$

9 $\quad\quad\quad$ **if** $\mathsf{c}(v) = 0$ **then** $\mathrm{Z} \leftarrow \mathrm{Z} \cup v$

10 $\quad$ **return** $(\mathrm{Z} \cap \mu_{\varrho}^{-1}(0), \mathrm{Z} \setminus \mu_{\varrho}^{-1}(0))$

Algorithm 23 simply computes the lift of all the positions contained in its input set N (Line 3) and then identifies the new positions that will be lifted by either the next application of $\mathsf{prg}_{\mathrm{o}}$, namely $\mathrm{Z} \cap \mu_{\varrho}^{-1}(0)$, or the subsequent application of prg_{+}, namely $\mathrm{Z} \setminus \mu_{\varrho}^{-1}(0)$. To do so, it first reinitializes the counter for the positions just lifted (Line 4) and, then, for each of their incoming moves (Line 5), verifies if there exists a new position whose measure needs to be increased. The case of an incoming 0-move is trivial (Lines 6-7). Therefore, let us consider the opponent player. A position $v \in \mathrm{Ps}_{1}$ needs to be lifted only if $\mu_{\varrho}(v) < \mu_{\varrho}(u) + v$, for all adjacents $u \in Mv(v)$. Therefore, we decrement the associated counter (Line 8) every time a non-progress move, that previously satisfied the progress condition *w.r.t.* the unlifted QDR, is identified. The counter reaching zero means that the above condition is satisfied, thus, the considered position need to be lifted in the next iteration (Line 9).

Algorithm 24: Efficient Quasi Dominion Operator

```
   signature Δ: 2^Ps → 2^Ps
   procedure Δ(N)
1      Q ← ∅
2      d ← ∅
3      while N ⊈ Q do
4          Q ← Q ∪ N
5          N ← pre(N)
6      return Q
   signature pre: 2^Ps → 2^Ps
   procedure pre(N)
7      Z ← ∅
8      d ← d[v ∈ (Mv^{-1}(N) ∩ Ps_⊟) \ dom(d) ↦ c(v)]
9      foreach (v, u) ∈ Mv; u ∈ N do
10         if v ∈ Ps_⊕ then
11             if σ_ϱ(v) = u then Z ← Z ∪ v
           else
12             if μ_ϱ(v) ≥ μ_ϱ(u) + v then d(v) ← d(v) − 1
13             if d(v) = 0 then Z ← Z ∪ v
14     return Z
```

Algorithm 24 computes the weak quasi dominion $\Delta(\varrho)$, starting from its trigger set $\mathrm{N} = \mathsf{npp}(\varrho)$ that contains all the non-progress positions in $\mathsf{Q}(\varrho)$. The implementation almost precisely follows the functional definition of the two operators Δ and pre, by caring only about keeping the whole computation cost linear in the number of incoming moves in each position contained in the resulting set. To do so, we exploit the same tricks used in the previous procedure employing a counter d for the 1-positions. Note that, d contains a copy of the values in c, in order to preserve the values in c for the other procedure.

Finally, Algorithm 25 implements the procedure described in Algorithm 21. It first computes the weak quasi dominion $\Delta(\varrho)$, by calling Algorithm 24 (Line 2). After that, it identifies its escape positions and the associated forfeit, in order to identify the set of best-escape positions that need to be lifted (Line 4). To do so, we employ a priority queue T based on a min-heap, which will contain at most S different forfeit values during the entire execution of the algorithm (positions associated with the same

forfeit are clustered together). Obviously, each insert, decrease-key, and remove-min operation on T will require time $O(\log S)$. The while-loop at Line 6 simulates the while-loop at Line 2 of Algorithm 21, where instructions at Lines 7-9 precisely correspond to those at Lines 3-5. After the measure update of the best-escape positions in E, the associated counters in c are reinitialized (line 10). At this point, an analysis on the incoming moves of E takes place (Line 11). For all moves $(v, u) \in Mv$ with $u \in \mathrm{E}$ and $v \notin \mathrm{Q}$, the algorithm performs, at Lines 17-22, almost exactly the same operations done by Algorithm 23 at Lines 6-9. The only difference here is that 1-positions can only be forced to lift their measure if they are not yet contained in the quasi dominion $\mathsf{Q}(\varrho)$. The case $v \in \mathrm{Q}$, instead, identifies a possible discovering of a new escape of the remaining weak quasi dominion (Line 12). If $v \in \mathrm{Ps}_1$, this is obviously an escape from Q, thus, it needs to be added to the priority queue T paired with the associated best-escape forfeit computed along the move (v, u) (Lines 13-14). If v is already contained in T, the associated valued is decreased, if necessary. The case $v \in \mathrm{Ps}_0$ is more complicated, since a 0-position is an escape if and only if its current strategy exits from Q and it has no move within Q that allows an increase of its measure. To do this check, once again, we employ the counter trick, where this time we associate with a 0-position in $\Delta(\varrho)$ the number of moves that satisfy the above property (Line 5). If the move (v, u) satisfies the property *w.r.t.* the unlifted QDR(*i.e.*, before the lifted of u occurred), then the corresponding counter $\mathsf{g}(v)$ is decreased (Line 15). When the counter reaches value 0, the position is necessarily an escape, so, it is added to the queue paired with its best possible forfeit (Line 16). Line 23 calls the win function in order to identify a possible new 0-dominion. Finally, Lines 24-29 update both the set of positions Z to be lifted in the next iteration and the counter c, by executing exactly the same instructions as those at Lines 18-22 on the moves that reach the dominion Q.

Algorithm 25: Efficient Progress Plus Operator

signature $\mathsf{prg}_+ \colon 2^{\mathrm{Ps}} \to 2^{\mathrm{Ps}} \times 2^{\mathrm{Ps}}$

procedure $\mathsf{prg}_+(\mathrm{N})$

$\mathrm{Z} \leftarrow \emptyset$

$\mathrm{Q} \leftarrow \Delta(\varrho, \mathsf{c}, \mathrm{N})$

$\widehat{\mu} \leftarrow \{v \in \mathrm{Q} \mapsto \mu_\varrho(v)\}$

$\mathrm{T} \leftarrow \{(v, \mathsf{bef}(\mu_\varrho, \mathrm{Q}, v)) \in \mathsf{esc}(\varrho, \mathrm{Q}) \times \mathbb{N}\}$

$\mathsf{g} \leftarrow \{v \in \mathrm{Q} \cap \mathrm{Ps}_\oplus \mapsto |\{u \in Mv(v) \cap \mathrm{Q} : \sigma_\varrho(v) = u \vee \mu_\varrho(v) < \mu_\varrho(u) + v\}|\}$

while $\mathrm{T} \neq \emptyset$ **do**

  $(\mathrm{E}, \mathrm{T}) \leftarrow \mathsf{extmin}(\mathrm{T})$

  $\varrho \leftarrow \mathsf{lift}(\varrho, \mathrm{E}, \overline{\mathrm{Q}})$

  $\mathrm{Q} \leftarrow \mathrm{Q} \setminus \mathrm{E}$

  $\mathsf{c} \leftarrow \mathsf{c}[v \in \mathrm{E} \cap \mathrm{Ps}_\boxminus \mapsto |\{u \in Mv(v) : \mu_\varrho(v) \geq \mu_\varrho(u) + v\}|]$

  foreach $(v, u) \in Mv$; $u \in \mathrm{E}$ **do**

    if $v \in \mathrm{Q}$ **then**

      if $v \in \mathrm{Ps}_\boxminus$ **then**

        $\mathrm{T} \leftarrow \mathrm{T} \cup (v, \mu_\varrho(u) + v - \mu_\varrho(v))$

      else

        if $\sigma_\varrho(v) = u \vee \mu_\varrho(v) < \widehat{\mu}(u) + v$ **then** $\mathsf{g}(v) \leftarrow \mathsf{g}(v) - 1$

        if $\mathsf{g}(v) = 0$ **then** $\mathrm{T} \leftarrow \mathrm{T} \cup (v, \mathsf{bef}(\mu_\varrho, \mathrm{Q}, v))$

    else if $\mu_\varrho(v) < \mu_\varrho(u) + v$ **then**

      if $v \in \mathrm{Ps}_\oplus$ **then**

        $\mathrm{Z} \leftarrow \mathrm{Z} \cup v$

      else if $\mu_\varrho(v) = 0$ **then**

        if $\widehat{\mu}(u) + v = 0$ **then** $\mathsf{c}(v) \leftarrow \mathsf{c}(v) - 1$

        if $\mathsf{c}(v) = 0$ **then** $\mathrm{Z} \leftarrow \mathrm{Z} \cup v$

$\varrho \leftarrow \mathsf{win}(\varrho, \mathrm{Q})$

foreach $(v, u) \in Mv$; $u \in \mathrm{Q}$ **do**

  if $v \in \mathrm{Ps}_\oplus$ **then**

    $\mathrm{Z} \leftarrow \mathrm{Z} \cup v$

  else if $\mu_\varrho(v) = 0$ **then**

    if $\widehat{\mu}(u) + v = 0$ **then** $\mathsf{c}(v) \leftarrow \mathsf{c}(v) - 1$

    if $\mathsf{c}(v) = 0$ **then** $\mathrm{Z} \leftarrow \mathrm{Z} \cup v$

return $(\mathrm{Z} \cap \mu_\varrho^{-1}(0), \mathrm{Z} \setminus \mu_\varrho^{-1}(0))$

□

9.4 Experimental Evaluation

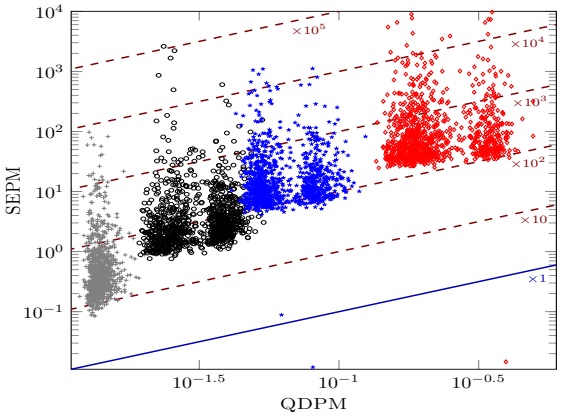


Figure 9.5: Experiments on random games with 5000 positions.

In order to assess the effectiveness of the proposed approach, we implemented QDPM, the Optimal Strategy Improvement algorithm (OSI [Sch08a]), and SEPM [BCD+11], the most (theoretical) efficient known solution to the problem and the more closely related one to QDPM, in `C++` within Oink [vD18]. Oink has been developed as a framework to compare parity game solvers. However, extending the framework to deal with MPGs is not difficult. The form of the arenas of the two types of games essentially coincide, the only relevant difference being that MPGs allow negative numbers to label game positions. We ran the solvers against randomly generated MPGs of various sizes. [1]

Figure 9.5 compares the solution time, expressed in seconds, of QDPM and SEPM on 4000 games, each with 5000 positions and randomly assigned weights in the range $[-15000, 15000]$. The scale of both axes is logarithmic. The experiments are divided in 4 clusters, each containing 1000 games. The benchmarks in different clusters differ in the maximal number m of outgoing moves per position, with $m \in \{10, 20, 40, 80\}$. These experiments clearly show that QDPM substantially outperforms SEPM. Most often, the gap between the two algorithms is between two and three orders of magnitude, as indicated by the dashed diagonal lines. It also shows that SEPM is particularly sensitive to the density of the underlying graph, as its performance degrades significantly as the number of moves increases. The maximal solution time was 8940 sec. for SEPM and 0.5 sec. for QDPM.

Figure 9.6 and 9.7, instead, compare the two algorithms fixing the maximal out-degree of the underlying graphs to 2, in the left-hand picture, and to 40, in the right-hand one, while increasing the number of positions from 10^3 to 10^5 along the x-axis. Each picture displays the performance results on 2800 games. Each point shows the total time to solve 100 randomly generated games with that given number of positions, which increases by 1000 up to size $2 \cdot 10^3$ and by 10000, thereafter. In both pictures

[1] The experiments were carried out on a 64-bit 3.9GHz quad-core machine, with Intel i5-6600K processor and 8GB of RAM, running Ubuntu 18.04.

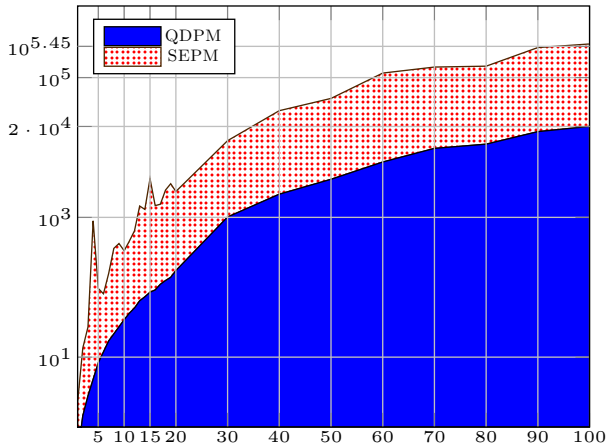


Figure 9.6: Total solution times in seconds of SEPM and QDPM on 5600 random games and maximal out-degree 2.

the scale is logarithmic. For the experiments in the right-hand picture we had to set a timeout for SEPM to 45 minutes per game, which was hit most of the times on the bigger ones.

Once again, the QDPM significantly outperforms SEPM on both kinds of benchmarks, with a gap of more than an order of magnitude on the first ones, and a gap of more than three orders of magnitude on the second ones. The results also confirm that the performance gap grows considerably as the number of moves per position increases.

We are not aware of actual concrete benchmarks for MPGs. However, exploiting the standard encoding of parity games into mean-payoff games [Jur98], we can compare the behavior of SEPM, QDPM and OSI on concrete verification problems encoded as parity games. For completeness, Table 9.1 reports some experiments on such problems.

The table reports the execution times, expressed in seconds, required by the algorithms to solve instances of two classic verification problems: the Elevator Verification and the Language Inclusion problems. These two benchmarks are included in the PGSolver [FL09] toolkit and are often used as benchmarks for parity games solvers. The first benchmark is a *verification under fairness* constraints of a simple model of an elevator, while the second one encodes the *language inclusion* problem between a non-deterministic Büchi automaton and a deterministic one. The results on various instances of those problems confirm that QDPM (and OSI as well) significantly outperforms the classic progress measure approach. Note also that the translation into MPGs, which encodes priorities as weights whose absolute value is exponential in the values of the priorities, leads to games with weights of high magnitude. Hence, the results in Table 9.1 provide further evidence that QDPM is far less dependent on the absolute value

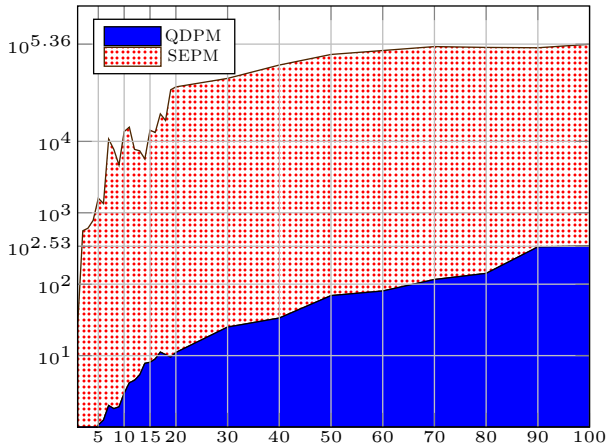


Figure 9.7: Total solution times in seconds of SEPM and QDPM on 5600 random games and maximal out-degree 40.

of the weights. They also show that QDPM can be very effective for the solution of real-world qualitative verification problems. It is worth noting, though, that the translation from parity to MPGs gives rise to weights that are exponentially distant from each other [Jur98]. As a consequence, the resulting benchmarks are not necessarily representative of MPGs, being a very restricted subclass. Nonetheless, they provide evidence of the applicability of the approach in practical scenarios.

Benchmark	Positions	Moves	SEPM	QDPM	OSI
Elevator 1	144	234	0.04	**0.0002**	**0.0002**
Elevator 2	564	950	8.80	0.0007	**0.0006**
Elevator 3	2688	4544	4675.71	0.0062	**0.0028**
Elevator 4	15683	26354	-	0.0528	**0.0379**
Lang. Incl. 1	170	1094	3.18	**0.0002**	0.0003
Lang. Incl. 2	304	1222	16.76	**0.0002**	0.0004
Lang. Incl. 3	428	878	20.25	**0.0002**	0.0004
Lang. Incl. 4	628	1538	135.51	**0.0003**	0.0006
Lang. Incl. 5	509	2126	148.37	**0.0003**	0.0005
Lang. Incl. 6	835	2914	834.90	**0.0005**	0.0010
Lang. Incl. 7	1658	4544	2277.87	**0.0002**	0.0009
Lang. Incl. 8	14578	17278	-	**0.0008**	0.0064
Lang. Incl. 9	25838	29438	-	**0.0015**	0.0137
Lang. Incl. 10	29874	34956	-	**0.0063**	0.0424

Table 9.1: Experiments on concrete verification problems.

Finally, Figure 9.8, compares QDPM and OSI on a smaller benchmark of 800 games with maximal

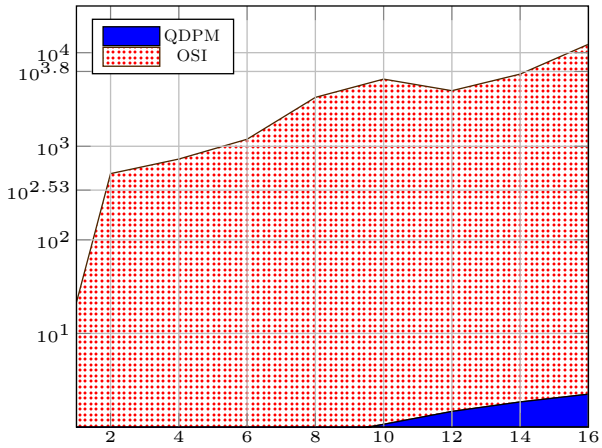


Figure 9.8: Total solution times in seconds of OSI and QDPM on 1000 random games and maximal out-degree 20.

out-degree 20 and randomly assigned weights in the range [-5000, 5000]. [2] Each point shows, in a logarithmic scale, the total time to solve 100 randomly generated games with the corresponding number of positions reported on the x-axis, which increases by 2000 up to size 16000. We set the timeout to 1200 seconds that has been reached 3 times by OSI solving the last cluster of games. Similarly to the experiments for SEPM, also in this case QDPM outperforms OSI, being orders of magnitude faster.

[2]The experiments were carried out on a 64-bit 1.6GHz quad-core machine, with Intel i5-8250U processor and 8GB of RAM, running Ubuntu 18.04.

Chapter 10

Conclusion

In this thesis, we considered the problem of solving *parity games* and *mean-payoff games*, two types of infinite-duration games over graphs having relevant applications in various branches of theoretical computer science.

We first presented a family of parity games that has been proved to be challenging for several solution approaches proposed in the literature [BDM17, BDM19a]. This family is resilient to effective heuristics, such as memoization and decomposition techniques, that, in some cases, can be used to break existing lower bounds. Specifically, we have shown that solving games in this family requires exponential time for the Recursive algorithm, regardless of whether it is endowed with memoization, SCC-decomposition or dominion decomposition techniques. It also provides a lower bound to progress measure based approaches. We have proved, indeed, that solving these games requires exponential time for Small Progress Measure [Jur00] and quasi-polynomial time for its succinct version [JL17], for which no lower bound was available before [CDF+19].

Spurred by these results, we have proposed a novel solution technique, based on the notion of *quasi dominions* and a *priority-promotion* mechanism. More precisely, three solution algorithms for parity games were presented and analyzed: *priority promotion* [BDM16c, BDM18b] (PP), *delayed promotion* [BDM16a, BDM18a] (DP), and *region recovery* [BDM16b] (RR). We gave proofs of their correctness and provided an accurate analysis of time and space complexities. For the PP algorithm, an exponential upper bound in the number of priorities is given and a lower bound is also presented in the form of a family of parity games on which the new technique exhibits an exponential behavior. On the bright side, it exhibits the best space complexity among the currently known algorithms for parity games, including the quasi-polynomial ones. In fact, we showed that the maximal additional space needed to

solve a parity game is linear in the number of positions, logarithmic in the number of priorities, and independent of the number of moves in the game (actually, it is logarithmic in the number of moves, but this dependence is subsumed by that of the number of positions). This is an important result, in particular considering that in practical applications we often need to deal with games having a very high number of positions, moves, and, in some cases, priorities. Therefore, low space requirements are essential for practical scalability. We also devised a quite general priority-promotion-based schema that can be instantiated with various game decomposition techniques, such as classic SCC-decomposition. We then investigated a finer priority-promotion policy in the DP algorithm with the aim to minimize the number of region resets after the occurrence of a priority promotion operation. This is done by delaying the promotions of a given priority as much as possible. Finally, with the RR algorithm, we presented a technique to recovery information from the regions that were reset after a promotion operation.

As an orthogonal contribution, we also proposed a new solution algorithm for mean-payoff games, by integrating the notions of progress measures and quasi dominions [BDM19b]. We argue that the integration of these two concepts offers a significant speed up in the convergence to the solution, at no additional computational cost *w.r.t.* the best known algorithm to date [BCD$^+$11]. This fact is supported by the existence of a family of games on which the combined approach can perform arbitrarily better than a classic progress measure based solution.

To assess the effectiveness of all the presented algorithms, experiments were conducted against both concrete and synthetic problems. We compared the new algorithms with the state-of-the-art parity games solvers, initially implemented in PGSolver [FL09] and, subsequently, in OINK [vD18] (the reported results for the parity solvers are those obtained for the PGSolverimplementation, since all experiments performed with OINK on the same benchmarks essentially confirm the analysis; the mean-payoff experiments were conducted in OINK, instead).

On parity games, the results were very positive, showing that the proposed approach is extremely effective in practice, outperforming all existing ones, including the quasi-polynomial algorithms, such as [FJS$^+$17] and [JL17]. Experiments comparing the vanilla PP algorithm and its optimizations via decomposition techniques also revealed that these techniques are extremely helpful in cutting down solution time, often of several orders of magnitude. Moreover, both DP and RR outperform PP, increasing the gap between the priority-promotion approaches and the state-of-the-art solvers known in the literature.

For mean-payoff games, the experimental results showed that the introduction of the notion of quasi dominions in the context of quantitative games can often reduce solution times up to three orders of magnitude, suggesting that this approach may be very effective in practical applications as well. We

believe, indeed, that the integration of techniques we devised is general enough to be applied to other types of games. In particular, the application of quasi dominions in conjunction with progress measure based approaches, such as those of [JL17] and [FJS+17], may lead to practically efficient quasi-polynomial algorithms for parity games.

List of Figures

List of Tables

List of Algorithms

Bibliography

[ABG14a] X. Allamigeon, P. Benchimol, and S. Gaubert. Combinatorial Simplex Algorithms Can Solve Mean-Payoff Games. *SIAM*, 24(4):2096–2117, 2014.

[ABG14b] X. Allamigeon, P. Benchimol, and S. Gaubert. The Tropical Shadow-Vertex Algorithm Solves Mean-Payoff Games in Polynomial Time on Average. In *ICALP'14*, pages 89–100, 2014.

[ABGJ15] X. Allamigeon, P. Benchimol, S. Gaubert, and M. Joswig. Tropicalizing the Simplex Algorithm. *SIAM*, 29(2):751–795, 2015.

[AHK02] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *AM*, 160(2):781–793, 2004.

[BBFR13] A. Bohy, V. Bruyère, E. Filiot, and J.-F. Raskin. Synthesis from LTL Specifications with Mean-Payoff Objectives. In *TACAS'13*, pages 169–184, 2013.

[BC12] L. Brim and J. Chaloupka. Using Strategy Improvement to Stay Alive. *IJFCS*, 23(3):585–608, 2012.

[BC18] M. Bojańczyk and W. Czerwiński. *An Automata Toolbox.* 2018. https://www.mimuw.edu.pl/ bojan/papers/toolbox-reduced-feb6.pdf.

[BCD+11] L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster Algorithms for Mean-Payoff Games. *FMSD*, 38(2):97–118, 2011.

[BCHK11] U. Boker, K. Chatterjee, T.A. Henzinger, and O. Kupferman. Temporal Specifications with Accumulative Values. In *LICS'11*, pages 43–52, 2011.

[BDHK06] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-Width and Parity Games. In *STACS'06*, LNCS 3884, pages 524–536. Springer, 2006.

[BDM16a] M. Benerecetti, D. Dell'Erba, and F. Mogavero. A Delayed Promotion Policy for Parity Games. In *GANDALF'16*, EPTCS 226, pages 30–45, 2016.

[BDM16b] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Improving Priority Promotion for Parity Games. In *HVC'16*, LNCS 10028, pages 1–17. Springer, 2016.

[BDM16c] 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.

[BDM17] 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.

[BDM18a] M. Benerecetti, D. Dell'Erba, and F. Mogavero. A Delayed Promotion Policy for Parity Games. *IC*, 262(2):221–240, 2018.

[BDM18b] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Parity Games via Priority Promotion. *FMSD*, 52(2):193–226, 2018.

[BDM19a] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Robust Worst Cases for Parity Games Algorithms. Accepted for publication on Information and Computation., 2019.

[BDM19b] M. Benerecetti, D. Dell'Erba, and F. Mogavero. Solving Mean-Payoff Games via Quasi Dominions. abs/1907.06264, 2019. Under submission.

[BFL+08] P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, and J. Srba. Infinite Runs in Weighted Timed Automata with Energy Constraints. In *FORMATS'2008*, pages 33–47. Springer, 2008.

[BG01] D. Berwanger and E. Grädel. Games and Model Checking for Guarded Logics. In *LPAR'01*, LNCS 2250, pages 70–84. Springer, 2001.

[BG04] D. Berwanger and E. Grädel. Fixed-Point Logics and Solitaire Games. *TCS*, 37(6):675–694, 2004.

[BGKR12] D. Berwanger, E. Grädel, L. Kaiser, and R. Rabinovich. Entanglement and the Complexity of Directed Graphs. *TCS*, 463:2–25, 2012.

[BMM13] M. Benerecetti, F. Mogavero, and A. Murano. Substructure Temporal Logic. In *LICS'13*, pages 368–377. IEEECS, 2013.

[BMM15] M. Benerecetti, F. Mogavero, and A. Murano. Reasoning About Substructures and Games. *TOCL*, 16(3):25:1–46, 2015.

[Bou49] N. Bourbaki. Sur le Théorème de Zorn. *AM*, 2(6):434–437, 1949.

[BSV04] H. Björklund, S. Sandberg, and S.G. Vorobyov. A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean-Payoff Games. In *MFCS'04*, pages 673–685, 2004.

[BV07] H. Björklund and S.G. Vorobyov. A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean-Payoff Games. *DAM*, 155(2):210–229, 2007.

[CDF+19] W. Czerwinski, L. Daviaud, N. Fijalkow, M. Jurdzinski, R. Lazic, and P. Parys. Universal Trees Grow Inside Separating Automata: Quasi-Polynomial Lower Bounds for Parity Games. In *SODA'19*, pages 2333–2349. SIAM, 2019.

[CHJ09] R. Bloem K. Chatterjee, T.A. Henzinger, and B. Jobstmannand. Better Quality in Synthesis Through Quantitative Objectives. In *CAV'09*, pages 140–156, 2009.

[CHP10] K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. *IC*, 208(6):677–693, 2010.

[CJK+17] 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.

[Con92] A. Condon. The Complexity of Stochastic Games. *IC*, 96(2):203–224, 1992.

[CPR17] C. Comin, R. Posenato, and R. Rizzi. Hyper Temporal Networks - A Tractable Generalization of Simple Temporal Networks and its Relation to Mean-Payoff Games. *Constraints*, 22(2), 2017.

[CR15] C. Comin and R. Rizzi. Dynamic Consistency of Conditional Simple Temporal Networks via Mean-Payoff Games: A Singly-Exponential Time DC-checking. In *TIME'15*, pages 19–28. IEEECS, 2015.

[CR17] C. Comin and R. Rizzi. Improved Pseudo-Polynomial Bound for the Value Problem and Optimal Strategy Synthesis in Mean-Payoff Games. *Algorithmica*, 77(4), 2017.

[DG06] V. Dhingra and S. Gaubert. How to Solve Large Scale Deterministic Games with Mean Payoff by Policy Iteration. In *VALUETOOLS'06*, page 12. ACM, 2006.

[EJ91] E.A. Emerson and C.S. Jutla. Tree Automata, muCalculus, and Determinacy. In *FOCS'91*, pages 368–377. IEEECS, 1991.

[EJS93] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On Model Checking for the muCalculus and its Fragments. In *CAV'93*, LNCS 697, pages 385–396. Springer, 1993.

[EJS01] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On Model Checking for the muCalculus and its Fragments. *TCS*, 258(1-2):491–522, 2001.

[EL86] E.A. Emerson and C.-L. Lei. Temporal Reasoning Under Generalized Fairness Constraints. In *STACS'86*, LNCS 210, pages 267–278. Springer, 1986.

[EM79] A. Ehrenfeucht and J. Mycielski. Positional Strategies for Mean Payoff Games. *IJGT*, 8(2), 1979.

[Fea10] J. Fearnley. Non-Oblivious Strategy Improvement. In *LPAR'10*, LNCS 6355, pages 212–230. Springer, 2010.

[FJS+17] 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.

[FK92a] M.R. Fellows and N. Koblitz. Self-Witnessing Polynomial-Time Complexity and Prime Factorization. In *CSCT'92*, pages 107–110. IEEECS, 1992.

[FK92b] M.R. Fellows and N. Koblitz. Self-Witnessing Polynomial-Time Complexity and Prime Factorization. *DCC*, 2(3):231–235, 1992.

[FL09] O. Friedmann and M. Lange. Solving Parity Games in Practice. In *ATVA'09*, LNCS 5799, pages 182–196. Springer, 2009.

[FL11] J. Fearnley and O. Lachish. Parity Games on Graphs with Medium Tree-Width. In *MFCS'11*, LNCS 6907, pages 303–314. Springer, 2011.

[Fri09] O. Friedmann. An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know It. In *LICS'09*, pages 145–156. IEEECS, 2009.

[Fri11] O. Friedmann. Recursive Algorithm for Parity Games Requires Exponential Time. *RAIROTIA*, 45(4):449–457, 2011.

[FS12] J. Fearnley and S. Schewe. Time and Parallelizability Results for Parity Games with Bounded Treewidth. In *ICALP'12*, LNCS 7392, pages 189–200. Springer, 2012.

[GKK88] V.A. Gurvich, A.V. Karzanov, and L.G. Khachivan. Cyclic Games and an Algorithm to Find Minimax Cycle Means in Directed Graphs. *USSRCMMP*, 28(5):85–91, 1988.

[JL17] M. Jurdziński and R. Lazic. Succinct Progress Measures for Solving Parity Games. In *LICS'17*, pages 1–9. ACM, 2017.

[JPZ06] M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. In *SODA'06*, pages 117–123. SIAM, 2006.

[JPZ08] M. Jurdziński, M. Paterson, and U. Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SJM*, 38(4):1519–1532, 2008.

[Jur98] M. Jurdziński. Deciding the Winner in Parity Games is in UP ∩ co-UP. *IPL*, 68(3):119–124, 1998.

[Jur00] M. Jurdziński. Small Progress Measures for Solving Parity Games. In *STACS'00*, LNCS 1770, pages 290–301. Springer, 2000.

[KK91] N. Klarlund and D. Kozen. Rabin Measures and Their Applications to Fairness and Automata Theory. In *LICS'91*, pages 256–265. IEEECS, 1991.

[Kla91] N. Klarlund. Progress Measures for Complementation of omega-Automata with Applications to Temporal Logic. In *FOCS'91*, pages 358–367. IEEECS, 1991.

[KV98] O. Kupferman and M.Y. Vardi. Weak Alternating Automata and Tree Automata Emptiness. In *STOC'98*, pages 224–233. ACM, 1998.

[Leh18] K. Lehtinen. A Modal mu Perspective on Solving Parity Games in Quasi-Polynomial Time. In *LICS'18*, pages 639–648. ACM & IEEECS, 2018.

[LP07] Y.M. Lifshits and D.S. Pavlov. Potential theory for mean payoff games. *JMS*, 145(3):4967–4974, 2007.

[LSW19] K. Lehtinen, S. Schewe, and D. Wojtczak. Improving the complexity of Parys' recursive algorithm. abs/1904.11810, 2019.

[Mar75] A.D. Martin. Borel Determinacy. *AM*, 102(2):363–371, 1975.

[Mar85] A.D. Martin. A Purely Inductive Proof of Borel Determinacy. In *SPM'82*, Recursion Theory., pages 303–308. AMS and ASL, 1985.

[McN93] R. McNaughton. Infinite Games Played on Finite Graphs. *APAL*, 65:149–184, 1993.

[MMPV12] F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. What Makes ATL* Decidable? A Decidable Fragment of Strategy Logic. In *CONCUR'12*, LNCS 7454, pages 193–208. Springer, 2012.

[MMPV14] F. Mogavero, A. Murano, G. Perelli, and M.Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *TOCL*, 15(4):34:1–42, 2014.

[MMV10] F. Mogavero, A. Murano, and M.Y. Vardi. Reasoning About Strategies. In *FSTTCS'10*, LIPIcs 8, pages 133–144. Leibniz-Zentrum fuer Informatik, 2010.

[Mos84] A.W. Mostowski. Regular Expressions for Infinite Trees and a Standard Form of Automata. In *SCT'84*, LNCS 208, pages 157–168. Springer, 1984.

[Mos91] A.W. Mostowski. Games with Forbidden Positions. Technical report, University of Gdańsk, Gdańsk, Poland, 1991.

[Obd03] J. Obdrzálek. Fast Mu-Calculus Model Checking when Tree-Width Is Bounded. In *CAV'03*, LNCS 2725, pages 80–92. Springer, 2003.

[Obd07] J. Obdrzálek. Clique-Width and Parity Games. In *CSL'07*, LNCS 4646, pages 54–68. Springer, 2007.

[Par19] P. Parys. Parity Games: Zielonka's Algorithm in Quasi-Polynomial Time. In *MFCS'19*, LIPIcs, pages 1–10. Leibniz-Zentrum fuer Informatik, 2019.

[Pis99] N.N. Pisaruk. Mean-Cost Cyclical Games. *MOR*, 24(4):817–828, 1999.

[Sch07] S. Schewe. Solving Parity Games in Big Steps. In *FSTTCS'07*, LNCS 4855, pages 449–460. Springer, 2007.

[Sch08a] S. Schewe. An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games. In *CSL'08*, LNCS 5213, pages 369–384. Springer, 2008.

[Sch08b] S. Schewe. ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008.

[Sch17] S. Schewe. Solving Parity Games in Big Steps. *JCSS*, 84:243–262, 2017.

[SF06] 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.

[Sti01] C. Stirling. *Modal and Temporal Properties of Processes.* Texts in Computer Science. Springer, 2001.

[vD18] T. van Dijk. Oink: an Implementation and Evaluation of Modern Parity Game Solvers. In *TACAS'18*, LNCS 10805, pages 291–308. Springer, 2018.

[vD19] T. van Dijk. A Parity Game Tale of Two Counters. In *GANDALF'19*, EPTCS 305, pages 107–122, 2019.

[VJ00] J. Vöge and M. Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *CAV'00*, LNCS 1855, pages 202–215. Springer, 2000.

[Wil01] T. Wilke. Alternating Tree Automata, Parity Games, and Modal muCalculus. *BBMS*, 8(2):359–391, 2001.

[Wit50] E. Witt. Beweisstudien zum Satz von M. Zorn. *MN*, 4(1-6):434–438, 1950.

[Zie98] W. Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *TCS*, 200(1-2):135–183, 1998.

[ZP96] U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *TCS*, 158(1-2):343–359, 1996.