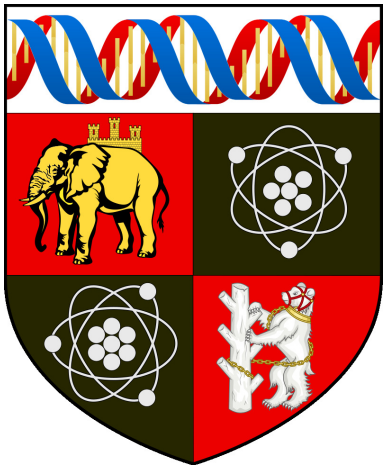


Attractor decompositions for solving parity and Rabin games

by

Thejaswini Kodaganalloor Srinivasa Raghavan

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy in Computer Science

Department of Computer Science

September 2023

Contents

List of Figures

Acknowledgments

This journey would not have been possible without the unwavering support, guidance, and contributions of numerous individuals who have shaped my academic pursuit.

I am profoundly grateful to my supervisor, Marcin Jurdziński, for his kindness, boundless patience, and the perfect balance of guidance and freedom he provided in navigating my research journey. His expertise, insightful feedback, and mentorship have been instrumental in shaping this thesis and nurturing my academic growth. His commitment to working meticulously through concepts, definitions, and proofs until they are distilled to their essential core has always inspired me. From his rigorous and scholarly approach to research, I have absorbed invaluable lessons that will undoubtedly guide my path in the future.

I extend my sincere appreciation to my examiners, Dmitry Chistikov and Sven Schewe, for examining my thesis and for providing detailed feedback and valuable insights into the work. Their thorough feedback greatly improved the quality of this work.

My thanks go to my doctoral advisors, Tom Gur and Dmitry Chistikov. In particular, Dmitry's exceptional dedication went beyond expectations, and his support across various facets of my research is deeply appreciated.

To my esteemed collaborators—Laure, Pierre, Rémi, Bala, Aditya, Irmak, Rupak, Michael, Antonio, Marcin, and Uéverton—I am indebted for the rich learning experiences shared and the insights gained from each collaboration. Thanks to Laure for our late-night writing session before my first deadline and to Bala, Aditya, and Irmak for showing how fun it is to work with a friend.

I wish to express my gratitude to Rupak Majumdar and Anne-Kathrin Schmuck for hosting me at MPI, enabling an environment conducive to collaboration. I also extend my thanks to Paul Gastin, Karoliina Lehtinen, and Stefan Schwoon, whose guidance during my PhD has been integral.

A special acknowledgement to some of my peers at Warwick with whom I have had insightful discussions about games and automata - Aditya Prakash, Amelia Chui, Henry Sinclair-Banks, and Alex Dixon.

Participation in Autoboz offered me an invaluable atmosphere for research, for which I extend my sincere appreciation.

The profound impact of my professors at CMI on my education cannot be overstated. They were instrumental in fostering my passion for mathematics, start-

ing from my days as an Undergrad and continuing through my Master's studies. Particular thanks to B. Srivatsan, C. Aishwarya, Narayana Kumar, and Madhavan Mukund for their guidance during my time at CMI.

I owe my early mathematical inspiration to Mr.Sadagopan Rajesh, whose guidance propelled my curiosity and thirst for learning.

Finally, my gratitude extends to those closest to my heart. To my friends from afar, including Bala, Nivedita, Neha, Ashutosh, Siva Tej, Bishal, and many others, who remained connected throughout the entirety of a challenging pandemic, I offer my heartfelt gratitude. To my friends from Warwick, Marcel, Matteo, Ninad, Namrata, Stas, Greg- your support and friendship have made this journey all the more meaningful. To my dearest Warwick friends, Iman and Aditya, with whom I share the joy of baking, I extend my heartfelt gratitude as they have been unwavering pillars of support, offering their assistance in various ways during the demanding final phases of my thesis journey. To Shruthi, who was just a Skype call away when I needed her the most. To Ahad, who has been my rock, and whose love and encouragement have been my constant strength throughout. To my parents and sister Manu, whose steadfast belief in me has been my guiding light.

Declarations

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified. Parts of the work in this thesis is from joint collaborations and is published or is being prepared for publication in related conferences. The following articles form parts of this thesis.

[DJT20] The Strahler Number of a Parity Game. Laure Daviaud, Marcin Jurdziński and K. S. Thejaswini. Published at ICALP 2020.
Chapters 4 and 5 contain results from this work.

[JMT22] Universal Algorithms for Parity Games and Nested Fixpoints. Marcin Jurdziński, Remi Morvan, and K. S. Thejaswini. Published at Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of his 60th Birthday 2022.
Chapter 3 contains results from Section 5 of this work.

[TOJ22] A Technique to Speed up Symmetric Attractor-Based Algorithms for Parity Games. K. S. Thejaswini, Pierre Ohlmann, and Marcin Jurdziński. Published at FSTTCS 2022.
Chapter 8 contains results from this work.

[MST23] Rabin Games and Colourful Universal Trees. Rupak Majumdar, Irmak Sağlam, and K. S. Thejaswini. Accepted to be published at TACAS 2024.
Chapters 9 and 10 contains results from this work.

[CPP+24] Simple and tight complexity lower bounds for solving Rabin games. Antonio Casares, Marcin Pilipczuk, Michał Pilipczuk, Uéverton S. Souza, and K. S. Thejaswini. Published at SOSA 2024.
Chapter 11 contains results from this work.

Results presented in this thesis in Chapter 6 and 7, not enumerated above, have been obtained under the supervision of Marcin Jurdziński and are being prepared for submission.

The following research was carried out in collaboration during the development of this thesis and have been published in conferences but does not form part of the thesis.

[BT21] Adaptive Synchronisation of Pushdown Automata. A. R. Balasubramanian and K. S. Thejaswini. CONCUR 2021.

[PT23] On History-Deterministic One-Counter Nets. Aditya Prakash and K. S. Thejaswini. FoSSaCS 2023.

[STSN24] Solving Two-Player Games Under Progress Assumptions. Anne-Kathrin Schmuck, K. S. Thejaswini, Irmak Saglam, and Satya Prakash Nayak. VMCAI 2024.

This work has not been submitted for any other degree or professional qualification.

Abstract

In this thesis, we consider the computational problem of deciding the winner in two player games on finite graphs with parity and Rabin objectives. Solving these games is a fundamental problem with applications to program verification, and synthesis, and it is closely linked to problems in automata theory and logic. We focus on understanding the structural properties of these games and devising algorithms that harness these properties.

At the core of this thesis is the concept of an attractor decomposition, a structured representation of a parity game that serves as a witness of winning for a player and that naturally corresponds to a tree. It has been established that universal trees—trees capable of embedding all possible trees emerging from an input parity game—play a pivotal role in serving as a search space for all known algorithms designed to solve parity games.

We define the Strahler number of a parity game as the smallest Strahler number of the tree of its attractor decomposition, and we establish that it functions as a robust and intuitive parameter. This concept propels us to construct succinct Strahler universal trees, enabling polynomial-time solutions for a wider range of parameter settings in parity games.

Through a relaxation of attractor decompositions that we call "decompositions," we formulate three novel algorithms for solving parity games. Our algorithms either boast a simpler description or faster runtime complexity in comparison to their predecessors, and they are designed to be easily adaptable to various universal trees, including our Strahler universal trees.

Finally, we extend the concept of attractor decompositions to "colourful decompositions," identified as witnesses of winning for Rabin games. The colourful trees that stem from colourful decompositions lead us to construct succinct colourful universal trees. This construction yields an algorithm which is an exponential improvement in space complexity and an exponential factor improvement in time complexity compared to other state-of-the-art algorithms.

Abbreviations

BFS Breadth First Search

ETH Exponential Time Hypothesis

FPT Fixed Parameter Tractable

LAR Latest Appearance Record

LTL Linear Temporal Logic

NFE Nested Fixpoint Equation

SCC Strongly Connected Component

Chapter 1

Introduction

Parity games. Parity games are two-player games played between Steven and Audrey on directed graphs, in which the ownership of these vertices is divided between the two players. The graphs on which these games are played are equipped with a natural number assigned to each vertex, called its priority. A token is placed at a designated start vertex. At each step, the owner of the vertex on which the token is currently placed chooses an outgoing edge. The token is then moved along the edge to the next vertex. The game proceeds for an infinite duration, creating a *play*: a countably infinite sequence of vertices seen by the token. Steven wins in a parity game if he can ensure, by choosing his edges appropriately, that for any play in the game, the highest priority visited infinitely often is even. Audrey wins otherwise.

Rabin games. Rabin games, like parity games, are also two-player games played on directed graphs, where each vertex is owned by one of Steven or Audrey. Each Rabin game has a finite set of colours. From this set of colours, for each vertex of the graph, a subset of the set of colours is assigned as its *good* colours and a subset of the set of colours is assigned as its *bad* colours. A token is placed on a designated start vertex, and the game proceeds similarly to that of a parity game, where the token is moved by the players along the edges, forming a play. Steven wins in a Rabin game if he can ensure that amongst the vertices that are visited infinitely often in a play, there is some colour that is a good colour for at least one vertex and is not a bad colour for any of these vertices.

1.1 The landscape

Parity and Rabin games have been studied since the late 1980s and they are arguably a fundamental model in automata theory and logic [EJ88, EJ91, Zie98, GTW02, BW18]. Applications of algorithms that solve such games include verification, program analysis, and synthesis. In particular, they are intimately linked to the problems of emptiness and complementation of non-deterministic automata on trees [EJ91, Zie98], model checking and satisfiability of fixpoint logics [EJS93, BW18], fair simulation relations [EWS01] and evaluation of nested fixpoint expressions [LBC$^+$94, BKMMP19, HS19]. Furthermore, Rabin conditions are suitable specifications for *general fairness constraints* [FK84] or to prove program termination under such constraints [KK91].

Automata and Logic. Automata, logic and games are inherently tied together with threads of expressivity and decidability results. The emptiness of tree automata or alternating automata with a parity or Rabin condition can be decided by solving parity or Rabin games derived directly from such automata. Moreover, the most effective translations from alternating parity or Rabin automata on infinite words to alternating weak automata have been inspired by algorithms for parity or Rabin games, respectively [KV98, BL19, DJL19].

The close connection between logic and automata has been explored since the 1960s [Bü62, Rab69]. We focus mainly on the modal μ-calculus here to highlight the connection of automata and games to logic. The modal μ-calculus has gained traction in the community since its introduction by Kozen [Koz83]. In terms of expressivity, this logic subsumes well-studied logics such as LTL [Pnu77] and CTL [CE81] (which are incomparable to each other in terms of their expressivity). It also subsumes the logic CTL* [EH86], which already subsumes both LTL and CTL.

Rabin showed that the Monadic Second Order logic (MSO) with n successors (SnS) is equi-expressive as Rabin tree automata [Rab69]. The result of Niwiński [Niw88, Niw97], followed by the work of Emerson and Jutla [EJ91], showed a tight (effectively translatable) equivalence in expressivity between the modal μ-calculus and parity tree automata.

Decision problems over the modal μ-calculus reduce to problems in automata theory and game theory [Str81, EJ91, MS95]. More specifically, the problem of checking if a Kripke structure is accepted by a given μ-calculus formula, also known as the model checking problem of modal μ-calculus, directly reduces to acceptance of a word by an alternating parity automata or, equivalently, deciding the winner

of a parity game [EJ91]. The problem of checking if a formula is satisfiable also reduces to the emptiness checking of non-deterministic Rabin automata on trees, or solving a parity or Rabin games thus obtained, but these reductions are rooted in fundamental results in automata theory [EJ91, Saf88, MS95]. This satisfiability problem is decidable and is also known to be EXPTIME-complete [SE89, EJ99].

For a more thorough view on modal μ-calculus and its relation to automata theory and games, we refer to the survey by Wilke [Wil01], and also a more recent survey by Hausmann and Piterman [HP22].

Model checking and Synthesis. The game-theoretic approach to system verification uses the theory of two player games on graphs to tackle verification and synthesis of systems, studied since the 1980s [CE81, QS83]. We give a brief overview on the interplay between the two player games we study and the areas of model checking and synthesis.

Model Checking. The model checking problem asks if one can construct machines that can, given a model and a specification, verify if the executions of this model satisfy the given specification. These specifications are usually expressed using temporal logical formulas. For our purposes, we limit our discussion to the model checking problem of modal μ-calculus, which asks if a given *Kripke structure* (labelled transition system) satisfies the properties expressed by a μ-calculus formula. As elaborated upon in our brief discussion on logic and games, the model checking problem reduces to the problem of solving parity games [EJ91]. Emerson, Jutla, and Sistla further showed that the model checking problem can be reduced to the non-emptiness problem of parity tree automata [EJS93]. It is also known that the model checking problem reduces to solving a system of *nested-fixpoint equations* [LBC$^+$94, Sei96, BW18]. The observation that model checking of modal μ-calculus can be done symbolically [McM93] is key to several industrial-scale model checkers [Wil01].

Synthesis. Posed by Church [Chu57] in the late 50s, the problem of synthesis asks if a reactive system can be automatically constructed from a logical specification. Due to the underlying connections between logic and games, the synthesis problem for several logics reduces to solving two-player games [BL69]. The synthesis problem, when the specification is given in LTL, is solved by converting such specifications into two-player games where the objective is assessed using a non-deterministic Büchi automaton over infinite words. These Büchi automata must be

determinised either to parity automata or Rabin automata [McN66, Saf88, MS95]. The solution to parity or Rabin games thus obtained effectively synthesises a controller.

The tree-automata based approach to synthesis was championed by Pneuli and Rosner [PR89] who showed 2-EXPTIME-completeness for LTL synthesis. Despite its discouraging complexity status, the problem of LTL synthesis is, in comparison, significantly more tractable than S1S synthesis (Monadic Second Order logic with one successor), which is non-elementary [Sto74]. We remark that the doubly exponential algorithm of Pneuli and Rosner was obtained by solving Rabin games. Fragments of LTL have also been considered while solving the synthesis problem, such as GR(1) (Generalised Reactivity (1)) proposed by Piterman, Pnueli, and Sa'ar [PPS06] for which the time taken by the synthesis problem is bounded by a single exponent.

Bloem, Chatterjee, and Jobstmann [BCJ18] provide a thorough overview of reactive synthesis where we redirect the curious reader.

Complexity status. The problem of deciding the winner of a parity game is in $\mathsf{NP} \cap \mathsf{coNP}$. It was also shown to be in the complexity class $\mathsf{UP} \cap \mathsf{coUP}$ by Jurdziński [Jur98]. The search version of the problem of finding strategies in a parity game for both players is in CLS [DP11], and therefore also in the complexity classes PPAD and PLS which contain CLS (see work of Fearnley et al. [FGHS21], for recent breakthrough result showing $\mathsf{PPAD} \cap \mathsf{PLS} = \mathsf{CLS}$). Rabin games, on the other hand, have been established to be NP complete in the work of Emerson and Jutla [EJ88]. The recent breakthrough of Calude, Jain, Khoussainov, Li, and Stephan [CJK$^+$22] showed that both parity and Rabin games are in FPT (fixed parameter tractable) for the parameter being the number of priorities or the number of colours, respectively. However, it has been open for over three decades whether solving parity games is in P.

Algorithmic efforts. For parity games and Rabin games, we use n to refer to the number of vertices and m the number of edges of the underlying graph on which these games are played. We use d to denote the number of distinct priorities in a parity game and k to denote the number of colours in a Rabin game.

Parity games. In their seminal work, Emerson and Jutla [EJ91] demonstrated the existence of *positional winning strategies* for both players in parity games. McNaughton [McN93], drawing inspiration from the contributions of Gurevich and

Harrington [GH82], as well as Yakhnis and Yakhnis [YY90], presented a recursive algorithm in his work to tackle a more general class of games, called Muller games. Building upon this foundation, Zielonka adapted McNaughton's algorithm, not only presenting an alternative proof of existence of positional winning strategies but also producing an algorithm with a running time of $\mathcal{O}\Big(\big(\frac{n}{d}\big)^d\Big)$ to solve parity games. An exponential space and $\mathcal{O}\Big(\big(\frac{n}{d}\big)^{d/2}\Big)$ time algorithm was given by Long, Browne, Clarke, Jha, and Marrero [LBC$^+$94] and later simplified by Seidl [Sei96] to solve model checking of modal μ-calclus—also known to be equivalent to solving parity games. Soon after, Jurdziński [Jur00] gave a small progress measure algorithm, which assigns a measure to the underlying game graph such that this measure is non-increasing along an edge in a play. This algorithm took time proportional to $\mathcal{O}\Big(\big(\frac{n}{d}\big)^{d/2}\Big)$, reducing the runtime by half in the exponent when compared to the McNaughton-Zielonka algorithm while maintaining closely its space complexity. Subsequently, Jurdziński, Paterson, and Zwick, using pre-processing that removed winning sets of fixed sizes, brought down the running time of the modified McNaughton-Zielonka algorithm to $n^{\mathcal{O}\left(\sqrt{n/\log n}\right)}$ [JPZ08]. Theirs was the first deterministic sub-exponential algorithm to match the expected runtime of the randomised algorithm to solve parity games by Björklund, Sandberg and Vorobyov [BSV03]. Inspired by this, Schewe [SF07, Sch17] gave an algorithm with the running time in $\mathcal{O}\Big(\big(\frac{n}{d^2}\big)^{d/3+0.5}\Big)$. For values of n and d where $d \in \mathcal{O}\big(\sqrt{n/\log n}\big)$, this algorithm outperformed the sub-exponential algorithm of Jurdziński, Paterson and Zwick, but theirs remained the state of the art for cases where d was asymptotically comparable to n.

Quasi-polynomial algorithms. A major breakthrough for algorithms to solve parity games came recently in 2017 when Calude, Jain, Khoussainov, Li and Stephan [CJK$^+$17, CJK$^+$22] provided a quasi-polynomial solution, along with a $\mathcal{O}(n^{\log d+6})$ upper bound of its running time. An algorithm is said to take quasi-polynomial running time if there is some constant c such that the running time of the algorithms is bounded by $n^{O(\log^c n)}$. This spurred research in parity games, leading to several illuminating results [GI17, BC17]. Following the algorithm of Calude et al., two independent algorithms emerged with significantly improved space complexity and closely matching runtime complexity. Jurdziński and Lazić's succinct progress measure algorithm [JL17] reinterpreted Jurdziński's small progress measure algorithm [Jur00] and encoded them succinctly. On the other hand, Fearnley, Jain, de Keijzer, Schewe, and Stephan's [FJdK$^+$19] algorithm modified Calude et

al.'s algorithm to significantly reduce the space complexity. The running time of the algorithm of Jurdziński and Lazic is $\mathcal{O}\big(\max\big\{\, 2^{\mathcal{O}(d \log d)}, \mathcal{O}(mn^{2.38}) \,\big\}\big)$ and it requires only quasi-linear space. The one by Fearnley et al. is also comparable in its runtime and space complexity. A closer modification resulted in a slight improvement was further obtained by Dell'Erba and Schewe using a modification of progress measures of Fearnley et al [DS22].

Subsequently, a pioneering new approach emerged in the form of Lehtinen's work, wherein she introduced a parameter, the "register number," aimed at solving parity games in quasi-polynomial time [Leh18, LB20] and in polynomial time when register number is bounded by a constant. However, the running time of the algorithm is bounded by $\mathcal{O}(nd^{k^2})$, where k is the register number of a parity game, which does not exceed $\lg(n) + 1$. Inspired by Bojańczyk and Czerwiński's [BC17] interpretation of the algortihm of Calude et al., Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić, and Parys [CDF$^+$19] exhibited a combinatorial structure of universal trees, provably underlying the techniques of Calude et al., of Jurdziński and Lazić, and of Lehtinen. An (n, h)-universal tree is an ordered tree that can *embed* in it any tree with at most n leaves and height h. Their results concluded that any algorithm that constructs a safety automaton whose accepting words work as a (specific kind of) *separator*, implicitly contains an $(n, d/2)$-universal tree. A separator is an automaton whose language separates the languages of words encoding plays that are (decisively) won by either Steven or Audrey in a parity game with n vertices and d priorities. Moreover, they showed that any $(n, d/2)$-universal tree must have size at least quasi-polynomial, thus providing evidence that the techniques developed in these papers may be insufficient for leading to further improvements in the complexity of solving parity games. However, we remark that such a combinatorial structure displayed underlying Lehtinen's algorithm does not have the same flavour as the other lower bound results. Indeed, her algorithm produces non-deterministic parity automata as separating automata as opposed to a deterministic safety automata of the others. The lower bound is obtained indirectly by arguing that the *safety automaton* derived from a *non-determinisitc parity automaton* with *some good-for-separation* properties has the lower bound induced by universal trees. Parys [Par20] later explained that this was sufficient, since the separators produced by Lehtinen's algorithm are what he called *suitable-for-parity-games* separators. He also gave an improved version of Lehtinen's algorithm. However, in his quest to achieve this improvement, he modified the definition of a register game to consider only positional strategies. Although his modified algorithm did have an improved running time compared to Lehtinen's algorithm, the state-space complexity still remained

quasi-polynomial, as opposed to state-of-the-art algorithms.

Focus soon shifted towards attractor-based algorithms since these lower bound techniques did not immediately seem applicable there. Attractors are the set of vertices from which one player has a reachability strategy to visit a target set of vertices. Algorithms that compute attractors include the McNaughton-Zielonka algorithm, which uses computing attractors as a primitive operation to find winning sets in the game. Parys [Par19] proposed an ingenious quasi-polynomial version of McNaughton-Zielonka algorithm, but Lehtinen, Schewe, and Wojtczak [LSW19], and Jurdziński and Morvan [JM20, JMT22] have again strongly linked all quasi-polynomial variants of these attractor-based algorithm to universal trees.

The work on universal trees has inspired several different directions of research. Motivated by the work of Czerwiński et al., an alternate formulation in terms of universal graphs was proposed, originally by Colcombet and Fijalkow [CF19], which has led to faster algorithms for mean-payoff games in the work of Colcombet, Fijalkow, Gawrychowski, and Ohlmann [FGO20, CFGO22]. The universal graphs perspective has also enabled a clear characterisation of the objectives for games on infinite graphs that have positional strategies, as demonstrated by Ohlmann [Ohl22]. Using these universal graphs and universal trees, algorithms to solve nested fixpoints have also evolved [HS19, ANP21]. In terms of automata theory, the most optimal translation of alternating parity automata on infinite words to alternating weak automata also uses the theory of universal trees [DJL19].

Rabin games. The problem of solving Rabin games was shown to be NP-complete by Emerson and Jutla [EJ88, EJ99] in the late 80s. In the same paper, Emerson and Jutla, and independently, Pnueli and Rosner [PR89], gave algorithms whose running times are $\mathcal{O}\big((nk)^{3k}\big)$ time, where n is the number of vertices of the game graph and k the number of colours. Algorithms to solve Rabin games have been tied intimately to algorithms that solve parity games. The arrival of the McNaughton-Zielonka [McN93, Zie98] algorithm meant that the fastest way to solve Rabin games was to convert them to a parity game using Latest Appearance Record (LAR) techniques [GH82] and then to use algorithms that solved parity games. But this changed within the decade when Kupferman and Vardi [KV98] reduced the cubic dependence on n^k to a quadratic one by showing that the non-emptiness of a Rabin tree automaton can be established in time $\mathcal{O}(mn^{2k}k!)$. Zielonka's algorithm works for more general conditions, which include Rabin conditions. His algorithm for Rabin games, in fact, runs in comparable time to the algorithm of Kupferman and Vardi. However, its running time was precisely established later by

Jurdziński [Jur00] to be $\mathcal{O}\big(mn^{2k}/(k/2)^k\big)$. Much later, Horn [Hor05] gave a different solution to solve Streett games (players' objectives are the opposite of that in Rabin games) with the same running time as Kupferman and Vardi's algorithm.

Inspired by the (then) fastest algorithm for parity games—the small progress measure algorithm by Jurdziński [Jur00]—Piterman and Pnueli [PP06] gave an $\mathcal{O}(mn^{k+1}kk!)$-time, $\mathcal{O}(nk)$-space algorithm to solve Rabin games, again improving algorithms that solve Rabin games by a factor of n^k.

The work of Piterman and Pnueli remained state-of-the-art for Rabin games until the quasi-polynomial breakthrough for parity games by Calude, Jain, Khoussainov, Li, and Stephan [CJK$^+$22]. The FPT algorithm of Calude et al. that solved parity games also gave an FPT algorithm for Rabin games where the parameter is the number of colours, as discussed briefly in their work. Converting a Rabin game to a parity game that preserves the winner leads to an exponentially large parity game with an increase in the number of vertices by a factor of $k!$ [GH82]. Nonetheless, we can solve Rabin games with n vertices, m edges, and k colours in time $\mathcal{O}\big(nmk!^{2+o(1)}\big)$ and $\mathcal{O}\big(nk!^{1+o(1)}\big)$ space, by solving this exponentially large parity game obtained using state-of-the-art algorithms [JL17, FJdK$^+$19]. This exact running time stems from using results in the work of Jurdziński and Lazić, or Fearnley et al., who reported a comprehensive analysis of the running time complexity of their algorithm for parity games across various parameter settings. However, it is important to note that the space requirements of all these FPT algorithms discussed so far are exponential.

Practical efforts. With a range of algorithms available to solve parity games, there have also been several implementations of these algorithms [BLV96, HKLN12, dAF07, BDM18]. While most implementations were sequential, there are now several multi-core implementations [vW08, vdB10, Fea17] designed to solve parity games, which make better use of machines with large memories and many CPUs.

However, dissatisfaction with the availability of practical implementations that match the theoretical advancements in solving parity games prompted Friedmann and Lange [FL09] to introduce a platform called PGSolver. This platform implemented several algorithms, facilitating comparisons among them across various families of parity games. In a bid to promote tools for addressing synthesis-related problems, SYNTCOMP, a synthesis competition was inaugurated in 2018 [JPA$^+$22]. Originally featuring three distinct tracks, each with specific synthesis specifications. Consequently, several tools emerged for solving parity games, including STRIX developed by Meyer, Sickert and Luttenberger [MSL18, LMS18] and Oink

by van Dijk [vD18, MSL18]. Notably, STRIX, which has consistently won the main SYNTCOMP competition from 2018 to 2021, uses parity game solvers to aid its LTL synthesis. Since 2021, there has been a dedicated track for parity game solvers [JPA+22].

1.2 Our contributions at a glance

A roadmap. This thesis consists of three parts, each part further contains three chapters.

Chapter 2 serves as a repository of definitions and algorithms, which we invoke throughout this thesis.

In Part I, comprising of Chapters 3 to 5, we understand the structure of parity games better in two ways. Initially, we tackle parity games that arise from nested fixpoint equations (Chapter 3) and provide algorithms to solve such games. Later, and more importantly, we define a fundamental parameter for parity games that we call the Strahler number of a parity game (Chapter 4), pivotal for characterising the games' intrinsic structure. We further show a combinatorial construction of Strahler universal trees which allow existing algorithms to solve parity games faster for various settings of different parameters, where one such parameter is its Strahler number (Chapter 5).

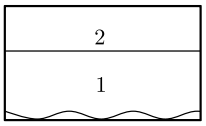
In Part II, which spans Chapters 6 to 8, we engineer three different algorithms: a strategy iteration algorithm (Chapter 6), an asymmetric attractor-based algorithm (Chapter 7) and a symmetric attractor-based algorithm (Chapter 8) to solve parity games. These algorithms are easily parameterized to run on any universal trees—the fundamental combinatorial object underlying all known algorithms that solve parity games.

Finally, in Part III, which includes Chapters 9 to 11, we give improved algorithms to solve Rabin games (Chapter 9). We extend our algorithm to also solve Rabin games with additional fairness constraints imposed on the opponent Audrey (Chapter 10). We show an alternate proof that our algorithms for Rabin games and their variations are optimal, conditional on the Exponential Time Hypothesis (Chapter 11).

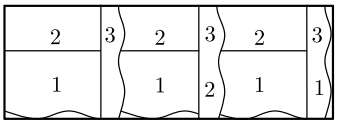
Subsequently in this chapter, we provide an intuitive overview of the attractor decomposition in a parity game and the associated trees. Using this understanding, we summarise the most important results of each chapter.

What are attractor decompositions? A *Steven dominion* is a subset of vertices of a parity game where Steven (has a strategy that) can ensure that all plays (that use this strategy) remain within this subset of vertices and are simultaneously winning for him. Every Steven dominion has a hierarchical subdivision of the set of vertices that is a structured encoding of a winning strategy for him. We call such subdivisions an *attractor decomposition*. Attractor decompositions are underlying in the work of McNaughton and of Zielonka [McN93, Zie98], and their connection to ordered trees has been made explicit in several works [DJL18, DJL19, JMT22].

Consider a parity game where the entire set of vertices is a Steven dominion and the priorities are assigned from the set $\{1, 2\}$. We know that a winning strategy of Steven must ensure that all plays visit the set of vertices of priority 2 infinitely often. Thus, every time he is at a vertex of priority 1, Steven's winning strategy ensures that he visits a vertex of priority 2. Therefore, the vertices of the game can be partitioned into two sets: vertices of priority 2 and vertices from which Steven can visit these "high" priority vertices within finitely many steps.

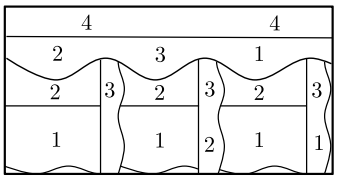


If we allow the range of priorities to extend from the set $\{1, 2\}$ to $\{1, 2, 3\}$ whilst still maintaining the condition that Steven is guaranteed to win from everywhere, Steven's strategies become more involved. Steven's winning strategy would enable him to partition the set of all vertices into "blocks," where within each block, the priority of the vertices is at most 2. Furthermore, in each block, he can employ his earlier format for winning, which is a strategy that leads the play to the set of vertices of priority 2 within that block. But such blocks can be arranged in order, with vertices of priority 3 (or lower) interspersed between such $\{1, 2\}$-blocks. Outside the $\{1, 2\}$-blocks, Steven's strategy would be to enter any $\{1, 2\}$-block arranged to the left. This way, Steven visits these vertices of priority 3 only in a transient fashion between these blocks, thereby seeing vertices of priority 3 only finitely often.



Further extending the range of priorities to $\{1, 2, 3, 4\}$ now alters the format of Steven's strategy as follows. If he can visit a vertex among the set of vertices of

priority 4, he does that. If not, then again follows a strategy similar to when the priorities were assigned from the set $\{1, 2, 3\}$. However, Steven can now visit these vertices of priority 3 multiple times, but between two such visits to the same vertex of priority 3, his strategy would ensure that he visits a priority 4. This strategy enables the game to be partitioned into vertices that can reach a priority 4 vertex in a finite number of steps, while the remaining part of the game consists of vertices with priorities in the set $\{1, 2, 3\}$. This remaining part of the game, in turn, can be subdivided, as discussed previously, with blocks of priority $\{1, 2\}$ and blocks of priority $\{1, 2, 3\}$, arranged linearly and alternating between the two.



This recursive decomposition can be extended to parity games of arbitrary priorities by decomposing the game in a similar manner recursively. Since strategies ensuring that Steven visits a fixed set of vertices are widely known as his attractor strategies, such decompositions of the games are termed attractor decompositions. These attractor decompositions are defined in a hierarchical manner and naturally correspond to trees. For example, the tree arising from the attractor decomposition represented above would be a tree of height one and with three leaves. If we had priorities from the set $\{1, \ldots, d\}$, then the associated tree would have height not more than $\lfloor d/2 \rfloor$. The observation that some of these partitions need to be non-empty, enforces that these trees have at most as many leaves as there are vertices in the game.

For a tree $\mathcal{T}$, a Steven dominion of a parity game is said to have a Steven $\mathcal{T}$-attractor decomposition if it has an attractor decomposition whose tree can be *embedded* in $\mathcal{T}$. A parity game has a Steven $\mathcal{T}$-attractor decomposition if the set of all vertices from which Steven can win—the largest Steven dominion—has a $\mathcal{T}$-attractor decomposition in the game. Since the tree of an attractor decomposition of a parity game with n vertices and d distinct priorities would have no more than n leaves and height $d/2$, such a parity game would therefore also have a $\mathcal{T}$-attractor decomposition where $\mathcal{T}$ is an $(n, d/2)$-universal tree—a tree that can embed all trees with n leaves and height $d/2$.

Part I

Part I's first chapter is Chapter 3, where we propose an algorithm that extends what was called the *universal algorithm* in the work of Jurdziński and Morvan [JM20]. We will refer to their algorithm as the Jurdziński-Morvan algorithm. We extend their algorithm, which originally was to solve parity games, to also solve nested fixpoint equations (thus making it even more universal!). We do so by using a characterisation of nested fixpoint equations over complete lattices called fixpoint games [BKMMP19, Ven08]. A system of nested fixpoint equations consists of d fixpoint equations, for i in $\{1, \ldots, d\}$, each of the form

$$X_i =_{\eta_i} f_i(X_1, \ldots, X_d),$$

where each η_i corresponds to the least or greatest fixpoint operator. A nested fixpoint equation can be converted to a nested fixpoint game, which is an exponentially large parity game. For equations over a powerset lattice of an n element set and with an alternation depth d in the system of equations, these parity games contain $\mathcal{O}\big(nd2^{nd}\big)$ many vertices and d distinct priorities. The winning vertices of Steven in a nested fixpoint game exactly identify the solution to the nested fixpoint equation from which this game arose. We show that although these parity games have exponentially many vertices, attractor-based algorithms can be modified to also work for nested fixpoint games, whilst keeping track of only polynomially many of the vertices of the fixpoint game at any given point of the algorithm.

We identify specific kinds of subgames of the fixpoint game and call these *flowery* subgames. We also show that attractors computed during the algorithm result in (specific kinds of) flowery subgames, and these subgames can also be complemented to result in flowery subgames within the algorithm. Moreover, flowery subgames can be represented succinctly. Since flowery subgames are sufficient to capture all subgames that arise during the computations in the Jurdziński-Morvan algorithm, this gives us a natural way to tweak their symmetric attractor-based algorithm to solve nested fixpoint equation over the powerset lattice. The Jurdziński-Morvan algorithms's recursive calls are dictated by two trees, and the correctness is guaranteed when these are both $(n, d/2)$-universal trees. Similar guarantees hold for the modified version that solves nested fixpoint games.

Theorem A. *The modified Jurdziński-Morvan algorithm that computes nested fixpoint equations takes quasi-polynomial time and polynomial space, when the underlying trees are quasi-polynomial sized universal trees.*

The rest of Part I is dedicated to our novel and arguably fundamental pa-

rameter of parity games, which we call the Strahler number. The Strahler number of a rooted tree is the largest height of a perfect binary tree that is its minor. We define the Strahler number of a parity game as the smallest of the Strahler numbers of the trees $\mathcal{T}$, such that the game has a $\mathcal{T}$-attractor decomposition.

Lehtinen's algorithm has avoided a tree based characterisation as it did not produce safety separating automata and instead produced parity separating automata. In her work, Lehtinen introduced the register number, a parameter of a parity game. We show in Theorem B that the Lehtinen number (which differs by at most 1 from the register number of a parity game) is exactly equal to the Strahler number of a parity game. The easier direction of proving that the Strahler number (of progress measures) bounds the Lehtinen number appears in the author's master's thesis [The19], and the question of whether these two values coincide was left open there. Chapter 4 introduces our definition of the Strahler number of a parity game. We demonstrate its connection to the Lehtinen number and provide a technical proof by constructing an attractor decomposition that shows the Strahler number is at most the register number.

Theorem B. *The Strahler number of a parity game is equal to its Lehtinen number.*

Recall that Lehtinen's algorithm required quasi-polynomial space and the runtime of her algorithm, although quasi-polynomial, did not match the state-of-the-art algorithms such as the ones by Jurdziński and Lazić. The concerns about the space requirements were also not resolved in the follow-up work of Parys [Par20]. We produce an algorithm which reduces the runtime of algorithms that solve parity games of a fixed Lehtinen number—or equivalently, the Strahler number—to match the state of the art. We define k-Strahler (n, h)-universal trees to be trees that can embed any ordered tree with n leaves, height h, and Strahler number k. Any (n, d)-small parity game whose Strahler number is at most k, can be solved by modifying the progress measure algorithm of Jurdziński and Lazić to instead run on Strahler universal trees. We construct optimal k-Strahler (n, h)-universal trees which ensures that the running time of our algorithm for solving parity games yields a novel trade-off

$$k \cdot \lg(d/k) = \mathcal{O}(\log n)$$

between the two natural parameters that measure the structural complexity of a parity game, which allows solving parity games in polynomial time. This includes as special cases the asymptotic settings of those parameters covered by the results of Calude, Jain, Khoussainov, Li, and Stephan [CJK+22], of Jurdziński and Lazić [JL17], and of Lehtinen and Boker [LB20], and it significantly extends the

range of such settings, for example to $d = 2^{\mathcal{O}\left(\sqrt{\log n}\right)}$ and $k = \mathcal{O}\left(\sqrt{\log n}\right)$.

Theorem C. *There is an algorithm for solving parity games with n vertices, d priorities, and of Strahler number k in quasi-linear space and time $n^{\mathcal{O}(1)} \cdot (d/2k)^k = n^{k \lg(d/k)/\lg n + \mathcal{O}(1)}$, which is polynomial in n if $k \cdot \lg(d/k) = \mathcal{O}(\log n)$.*

Part II

Having constructed small Strahler universal trees, we engineer algorithms in Part II to solve parity games that exploit the existence of these *small* trees.

We produce three different algorithms. A strategy iteration algorithm (Chapter 6), an asymmetric algorithm that produces an attractor decomposition for one player (Chapter 7), and finally, a symmetric algorithm that recursively solves games for both players symmetrically and exploits the progress made in a recursive subcall for one player to aid the other player (Chapter 8).

The underlying concept that is used for each of these algorithms is that of a *decomposition* of a parity game. Decompositions are a relaxation of attractor decompositions endowed with a partial order among them. Each of these algorithms take time that is linear in the size of the tree and at most polynomial space. Plugging in our succinct Strahler universal or the universal trees in the work of Jurdziński and Lazić, we ensure that these algorithms are comparable to the state of the art.

Strategy iteration. Strategy improvement algorithms form a class of algorithms used to solve two-player games with positional winning strategies [How60, VJ00, Lut08, Sch08]. Usually, strategy improvement algorithms start from an arbitrary positional strategy for one player. These algorithms use an underlying valuation that evaluates how good each strategy is. The strategies are then improved based on this valuation until an optimal strategy is found. Although strategy improvement algorithms can take exponential time in theory [Fri09, Fea10], in practice, they terminate very quickly. Koh and Loho [KL22] gave a new take on strategy improvement algorithms, which we refer to as strategy iteration algorithms. Their valuation of each strategy was based on both a fixed strategy and a progress measure that is maintained throughout the algorithm. For progress measures based on the universal trees of Jurdziński and Lazić and our Strahler universal trees, Koh and Loho's algorithm improves a strategy and computes the valuation of the strategy by computing a new progress measure in polynomial time. Moreover, the number of iterations required is linear in the size of the universal tree used, thereby giving a quasi-polynomial strategy iteration algorithm. The algorithm that computes this

improved progress measure is quite involved and uses regularity of the subtrees of the constructed universal trees.

We propose a strategy iteration algorithm that uses both a strategy and a decomposition to compute a valuation. We show that our algorithm based on decompositions, instead of progress measures, is simpler and we give an improved $\widetilde{\mathcal{O}}(|\mathcal{G}|d)$ upper bound on the time taken to perform each step of the strategy iteration for a parity game $\mathcal{G}$ with d distinct priorities. This is in contrast to Koh and Loho's algorithm for which each iteration takes time up to $\widetilde{\mathcal{O}}(|\mathcal{G}|^3)$ for quasi-polynomial universal trees. Moreover, our algorithm extends naturally to any tree rather than specifically constructed universal trees, and therefore terminates in time that is linear in the size of any fixed tree.

Theorem D. *For a parity game $\mathcal{G}$ with n vertices, d priorities, and a tree $\mathcal{T}$ of even level d, each iteration of the* strategy iteration *algorithm (Algorithm 2 on page 91) takes time $\widetilde{\mathcal{O}}(|\mathcal{G}|d)$. The valuation (of the decomposition and strategy maintained) at each step is strictly improving. The algorithm terminates with a $\mathcal{T}$-attractor decomposition of $\mathcal{G}$ within $n^2|\mathcal{T}|$ iterations.*

Asymmetric attractor-based algorithm. Algorithms that repeatedly compute attractors such as McNaughton-Zielonka [McN93, Zie98] and its several variants [BDM18, BDM$^+$21] outperform numerous algorithms that boast better theoretical complexities. We present an algorithm that iteratively calculates attractors, with its recursive calls being guided by a universal tree. However, it has two sharp points of contrast to other attractor-based algorithms whose recursive calls are guided by trees. Firstly, our algorithm is asymmetric and depends only on the tree for one player, in contrast to other attractor-based algorithms (McNaughton-Zielonka [McN93, Zie98], Jurdziński-Morvan [JMT22], or Lehtinen et al. [LPSW22]), which work recursively based on trees for both players. Secondly, our algorithm takes time only linear in the size of the tree it depends on, as opposed to the quadratic dependence of the current symmetric algorithms on the same. This improvement is achieved by using decompositions in our recursive attractor-based algorithm. The decompositions are used to preserve the progress made in each recursive subcall, thus leading to our algorithm, whose guarantees are stated below.

Theorem E. *For a parity game $\mathcal{G}$ and a tree $\mathcal{T}$, Algorithm 6 (on page 106) takes time at most linear in the number of nodes in $\mathcal{T}$ and polynomial in the size of the game $\mathcal{G}$ to produce the largest Steven dominion that has a Steven $\mathcal{T}$-attractor decomposition.*

Symmetric attractor-based algorithm. Although the previous algorithm has running time comparable to state-of-the-art algorithms, it is an algorithm that treats its players asymmetrically. This asymmetric treatment renders the algorithm intractable for some easy families of games as the worst-case running times are realised by the asymmetric algorithm on such families of games. For these examples, symmetric algorithms would have solved the same family of games in polynomial time. This motivates us to leverage the symmetric nature of the players in these games to build a recursive, symmetric, attractor-based algorithm, which matches the theoretical guarantees of the previous algorithm. We extend the techniques developed in Chapter 7 and show that even in recursive symmetric algorithms, the progress made in recursive calls can be encoded in the form of a decomposition.

Our symmetric algorithm is described in a way that it shows that its recursive calls are guided by trees similar to that of the other symmetric attractor-based algorithms. Therefore, our algorithm can be seen as a way to enhance the algorithms of Parys [Par19], the algorithm of Lehtinen, Schewe and Wojtczak [LSW19, LPSW22], as well as the algorithm of Jurdziński and Morvan [JMT22] to closely match (up to a polynomial factor) the running time of state-of-the-art algorithms [JL17, DJT20, FJdK$^+$19]. While the other symmetric attractor-based algorithms discard the progress made in their preceding recursive calls, we instead utilise it by a robust encoding of this progress in the form of a decomposition for *both players*, which in turn enables subsequent recursive calls to be made on smaller games, thereby improving our running times.

Theorem F. *For a parity game $\mathcal{G}$ and two trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$, the procedure* Univ-Even-Fast *(resp.* Univ-Odd-Fast*) in Algorithm 9 (on page 140) takes time $n^{\mathcal{O}(1)} \cdot \mathcal{O}\big(\max\big(|\mathcal{T}^{\text{Odd}}|, |\mathcal{T}^{\text{Even}}|\big)\big)$ to identify a set of vertices that includes all Steven dominia of $\mathcal{G}$ with a $\mathcal{T}^{\text{Even}}$-attractor decomposition and does not intersect with any Audrey dominia with a $\mathcal{T}^{\text{Odd}}$-attractor decomposition.*

For parity games with n vertices and d priorities, and two trees that are $(n, d/2)$-universal, these algorithms correctly identify the winning regions of the game $\mathcal{G}$ in time proportional to the size of an $(n, d/2)$-universal tree. In Chapter 8, we demonstrate the effectiveness of our symmetric algorithm with the following observation. Using exponentially large $(n, d/2)$-trees for both players, our symmetric algorithm solves several families of games in polynomial time whereas the McNaughton-Zielonka algorithm takes exponential time. Since the McNaughton-Zielonka algorithm's recursive calls are governed by the same exponentially large $(n, d/2)$-trees, we assert that our enhancement of the McNaughton and Zielonka algorithm significantly improves runtime for these specific game families.

Part III

Rabin games. We turn our attention to Rabin games. These games can encode parity games without a blow-up. However, deciding the winner in a Rabin game is NP-complete. Recall that the quasi-polynomial breakthrough of Calude et al. [CJK$^+$22] and its several follow-ups [JL17, FJdK$^+$19] gave algorithms to also solve Rabin games in time $\mathcal{O}\big(mn^2(k!)^{2+o(1)}\big)$ but with an exponential space requirement. We produce an algorithm that improves the dependence on $k!$ from $k!^{2+o(1)}$—obtained by converting a Rabin game into a parity game—to a $k!^{1+o(1)}$ factor in the running time while simultaneously improving its exponential space requirement to a polynomial one. Our main technical ingredient is a characterisation of winning strategies for Rabin games using *colourful decompositions*, an extension of attractor decompositions. Colourful decompositions have a recursive structure that can be captured by colourful ordered trees—an extension of ordered trees. We extend the concept of universality to such colourful trees and, using a combinatorial construction inspired by the universal trees implicitly constructed in the work of Jurdziński and Lazić, build succinct *colourful universal trees*. Our colourful universal trees are generalisations of universal trees in the work of Jurdziński and Lazić [JL17] to solve parity games, as well as pointer trees that appear in the work of Klarlund and Kozen [KK91].

Theorem G. *A winning strategy for Steven in a Rabin game with n vertices, m edges, and k colours can found using $\mathcal{O}(nk \log k \log n)$ space and time*

$$\widetilde{\mathcal{O}}\left(nm \cdot k! \min\left\{n2^k, \binom{\lceil \lg n \rceil + k}{k-1}\right\}\right).$$

Fair Rabin games. We later consider a modified version of Rabin games, called fair Rabin games. These games are played in arenas similar to Rabin games, but Audrey must be *fair* with respect to some specified *live* edges and also ensure that the Rabin condition is not satisfied. A play of Audrey is said to be fair if among these specified live edges, any edge's source being seen infinitely often also implies that this edge is taken infinitely often. Although such conditions can be encoded as a Rabin condition, this leads to an increase in the number of colours used to encode. We give a characterisation of such games similar to that of colourful decompositions. Steven has a winning strategy in a fair Rabin game if and only if he has a *fair colourful decomposition* of the vertices. These fair colourful decompositions also have the recursive structure of a colourful tree. Using our construction of colourful universal trees, we also obtain an algorithm that solves fair Rabin games, whose

running time is upper bounded by the same function that we gave for algorithms that solve Rabin games. Since identifying if Steven wins almost surely in stochastic Rabin games reduces (in log space) to finding a winner in a fair Rabin game, we also obtain algorithms for it.

Theorem H. *Finding the winner in a fair Rabin game can be determined in* $\mathcal{O}(nk \log n \log k)$ *space and time*

$$\widetilde{\mathcal{O}}\left(nm \cdot k! \left(\min\left\{n2^k, \binom{\ell + k}{k-1}\right\}\right)\right).$$

Finally, in Chapter 11, we give an alternate proof of a lower bound result for solving Rabin games. It was known from the work of Calude et al.[CJK$^+$22], that assuming the Exponential Time Hypothesis (ETH, the assumption that there exists $\delta > 0$ such that the 3SAT problem cannot be solved in time $\mathcal{O}(2^{\delta n})$), Rabin games cannot be solved by algorithms whose running time is bounded by $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$. Since our algorithm closely matches this bound, this result implies that our algorithm to solve Rabin games cannot be improved under the ETH.

Our alternate proof of this lower bound stems from a reduction from the problem called Permutation SAT. We show that specific kinds of instances of Permutation SAT do not have a $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$ time algorithm (where k is the number of variables and n is the number of clauses) under the ETH. Consequently, we reduce such instances of Permutation SAT to a Rabin game. Our reduction, we believe, is simpler, reminiscent of the NP-hardness reduction from Emerson and Jutla [EJ99], and more importantly, highlights the insight of the $k!$ factor in our algorithm better than the existing lower bound proof of Calude et al. which uses the problem of Dominating Set to prove the same.

Theorem I ([CJK$^+$22])**.** *Assuming the Exponential Time Hypothesis, there is no algorithm that solves Rabin games with* n *vertices and with* k *colours in time* $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$.

Chapter 2

Games, trees, and attractor decompositions

In this chapter, we establish notation that we use throughout the rest of the thesis. Most importantly, we define formally a parity game, and also demonstrate algorithms that solve a given parity game. We restrict ourselves mostly to presenting definitions and algorithms for parity games and we only define Rabin games. Concepts and notations that appear in only one or two chapters are defined closer to where they are needed.

Throughout, we use $\mathbb{N}$ to denote natural numbers $\{0, 1, 2, \dots\}$. We also use $\mathbb{N}_+$ to denote the positive natural numbers $\{1, 2, \dots\}$. For two natural numbers i and j, we denote the set of integers $\{x \in \mathbb{N} \mid i \leqslant x \leqslant j\}$ with $\{i, \dots, j\}$ and sometimes we write $[i, j]$ and just $[j]$ to denote $[1, j]$. For a positive natural number n, we write $\lg n$ to denote the binary logarithm, $\log_2(n)$. We use $\ln n$ for the natural logarithm of n, $\log_e(n)$. For a set X, we write $\mathcal{P}(X)$ to represent its powerset, the set consisting of all subsets of X.

A directed graph (V, E) is a finite set V of *vertices* and a set of ordered pairs of vertices E, referred to as its edges. We say that a vertex v is a neighbour of u if $(u, v) \in E$. Sometimes, we write $u \to v$ to mean $(u, v) \in E$. An undirected graph is a directed graph, except E is also required to be a symmetric relation. A path on a directed graph is a sequence of vertices such that any two consecutive vertices in the sequence belong to the set of edges. We say finite or infinite paths to refer to the finite or infinite sequences of vertices that define this path. A cycle is a finite path in this graph whose start and end vertices in the sequence are the same. A simple cycle is a cycle where, other than the start and the end vertices in the path, no two vertices are the same. When we refer to graphs, we mean directed graphs

unless explicitly mentioned otherwise.

Parity games. *Parity games* [EJ91] are zero-sum, two-player games played between Steven and Audrey. A specific instance of the game $\mathcal{G}$ consists of a directed graph (V, E) that is sink-free (where every vertex has an outgoing edge), a partition $(V_{\text{Even}}, V_{\text{Odd}})$ of the set of vertices V, a distinguished start vertex from the set of vertices, and a *priority function* $\pi : V \to \{0, 1, \ldots, d\}$ that assigns every vertex $v \in V$ with a natural number $\pi(v)$, called its *priority*. We say Steven *owns* vertices in V_{Even} and Audrey owns vertices in V_{Odd}. We say that a parity game is (n, d)-*small* if it has at most n vertices and the priorities of its vertices are bounded by d.

Strategies, Traps, and Dominions. For a parity game $\mathcal{G}$ as with the graoh (V, E), a Steven *strategy* is a function from the set of all finite paths ending at a Steven vertex on the graph (V, E) to a neighbour of this vertex. An infinite path starting at the start vertex in the underlying graph is said to be a *play*. A play $v_0, v_1, \ldots, v_i, \ldots$ is said to respect a strategy if for every vertex v_i that belongs to Steven, the vertex v_{i+1} is the one proposed by the strategy on the finite prefix of this play ending at v_i. An infinite play is said to be *even* if the highest priority of the vertices visited infinitely often is even, and otherwise it is said to be *odd*. We also say that a play is winning for Steven if it is even. Otherwise, the play is winning for Audrey. A game is said to be winning for Steven if he has a strategy such that *every* infinite play respecting the strategy is even. Such a strategy is often referred to as *Steven's winning strategy*. We know that parity games are *determined*, that is, a game is winning for either Steven or Audrey [Mar75].

Furthermore, the positional determinacy theorem for parity games states that if Steven has a winning strategy, then he can win by using strategies that do not require him to "remember the past". A *positional Steven strategy* is defined as a set $\sigma \subseteq E$ of edges such that:

- for every v owned by Steven, there is an edge $(v, u) \in \sigma$,
- for every v owned by Audrey, if $(v, u) \in E$ then $(v, u) \in \sigma$.

We say that σ is an even positional winning strategy if all infinite paths in the restricted graph (V, σ) are winning for him. Audrey positional strategies and positional winning strategies are defined similarly.

Theorem 2.0.1 (Positional determinacy of parity games [EJ91])**.** *If a game $\mathcal{G}$ is winning for Steven, then Steven has a positional winning strategy. The same holds for Audrey.*

When talking about strategies in parity games, for the most part, since both players have positional winning strategies, it is sufficient to verify the parity criterion on all (simple) cycles in the graph restricting to this strategy. More specifically, a parity game is winning for Steven if and only there is a positional winning strategy for Steven such that all simple cycles in the graph obtained from restricted to this strategy contains are even. Here, we say that a simple cycle in a parity game is even when the highest priority occurring in such a cycle is even. Otherwise, we call a simple cycle odd. However, we explicitly consider the parity criterion on infinite plays obtained from such a strategy when we find it more convenient.

Henceforth, in the context of parity games, when we refer to strategies, we usually mean positional ones. For a set S of vertices, we write $\mathcal{G} \cap S$ for the substructure of $\mathcal{G}$ whose graph is the subgraph of (V, E) induced by the sets of vertices S. Sometimes, we also write $\mathcal{G} \setminus S$ to denote $\mathcal{G} \cap (V \setminus S)$. We assume throughout that every vertex has at least one outgoing edge, and we reserve the term *subgame* to substructures $\mathcal{G} \cap S$, such that every vertex in the subgraph of (V, E) induced by S has at least one outgoing edge. For a subgame $\mathcal{G}' = \mathcal{G} \cap S$, we sometimes write $V(\mathcal{G}')$ for the set of vertices S that the subgame $\mathcal{G}'$ is induced by. But mostly, when there is no risk of confusion, we simply write $\mathcal{G}'$ instead of $V(\mathcal{G}')$.

For a non-empty set of vertices R, we say that a Steven strategy σ *traps Audrey in* R if $w \in R$ and $(w, u) \in \sigma$ imply $u \in R$. We say that a set of vertices R is a *trap for Audrey* [Zie98] if there is a Steven strategy that traps Audrey in R. Observe that if R is a trap in a game $\mathcal{G}$ then $\mathcal{G} \cap R$ is a subgame of $\mathcal{G}$. For brevity, we sometimes say that a subgame $\mathcal{G}'$ is a trap if $\mathcal{G}' = \mathcal{G} \cap T$ and the set T is a trap in $\mathcal{G}$. Moreover, the following property holds: if T is a trap for Steven in game $\mathcal{G}$ and T' is a trap for Steven in subgame $\mathcal{G} \cap T$ then T' is also a trap for Steven in $\mathcal{G}$. For a set of vertices $D \subseteq V$, we say that a Steven strategy σ is a *Steven dominion strategy on* D if σ traps Audrey in D and all paths in the subgraph (D, σ) are winning for Steven. Finally, we say that a set D of vertices is a *Steven dominion* [JPZ08] if there is a Steven dominion strategy on it.

Similarly, by swapping the roles of the two players, all concepts defined for Steven in a parity game are also defined for Audrey. We note that the sets of Steven dominions and of Audrey dominions are both closed under union, and hence the largest Steven and Audrey dominions exist, and they are the unions of all Steven and Audrey dominions, respectively. Moreover, every Steven dominion is disjoint from every Audrey dominion.

Reachability strategies and attractors. In a parity game $\mathcal{G}$, for a target set of vertices B and a set of vertices A such that $B \subseteq A$, we say that a Steven strategy σ is a *Steven reachability strategy to B from A* if every infinite path (including the starting vertex of the path) in the subgraph (V, σ) that starts from a vertex in A contains at least one vertex in B.

For every target set B, there is the largest set (with respect to set inclusion) A from which there is a Steven reachability strategy to B in $\mathcal{G}$; we call this set the *Steven attractor to B in $\mathcal{G}$* [Zie98]. We further say the set $A \setminus B$ is the strict Steven attractor to B in $V(\mathcal{G})$. *Audrey reachability strategies* and *Audrey attractors* are defined analogously. We highlight the simple fact that if A is an attractor for a player in $\mathcal{G}$ then its complement $V \setminus A$ is a trap for them and that attractors are monotone operators: if $B' \subseteq B$ then the attractor to B' is included in the attractor to B.

We define an attractor decomposition below that captures the intuition outlined in the introduction of this thesis.

If $\mathcal{G}$ is a parity game in which all priorities do not exceed a non-negative even number d then we say that

$$\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell) \rangle$$

is a *Steven d-attractor decomposition* [DJL18, DJL19] of $\mathcal{G}$ if:

1. A is the Steven attractor to the (possibly empty) set of vertices of priority d in $\mathcal{G}$;

and setting $\mathcal{G}_1 = \mathcal{G} \setminus A$, for all $i = 1, 2, \ldots, \ell$, we have:

2. S_i is a non-empty trap for Audrey in $\mathcal{G}_i$ in which every vertex priority is at most $d - 2$;
3. $\mathcal{A}_i$ is a Steven $(d-2)$-attractor decomposition of subgame $\mathcal{G} \cap S_i$;
4. A_i is the Steven attractor to S_i in $\mathcal{G}_i$;
5. $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$;

and the game $\mathcal{G}_{\ell+1}$ is empty. If $d = 0$ then we require that $\ell = 0$.

We deviate slightly from this definition of an attractor decomposition in Part II, in which we restate the modified definition to suit our algorithms.

The following proposition states that if a subgame induced by a trap for Audrey has a Steven attractor decomposition then the trap is a Steven dominion.

Indeed, a routine proof argues that the union of all the Steven reachability strategies, implicit in the attractors listed in the decomposition, is a Steven dominion strategy.

Proposition 2.0.2 ([Zie98, DJL18]). *If d is even, R is a trap for Audrey in $\mathcal{G}$, and there is a Steven d-attractor decomposition of $\mathcal{G} \cap R$, then R is a Steven dominion in $\mathcal{G}$.*

Attractor decompositions for Audrey can be defined in the analogous way by swapping the roles of players as expected, and then a dual version of the proposition holds by symmetry.

The following theorem implies that every vertex in a parity game is either in the largest Steven dominion or in the largest Audrey dominion—it is often referred to as the *positional determinacy theorem* for parity games.

Theorem 2.0.3 ([EJ91, McN93, Zie98]). *For every parity game $\mathcal{G}$, there is a partition of the set of vertices into a trap for Audrey W_{Even} and a trap for Steven W_{Odd}, such that there is a Steven attractor decomposition of $\mathcal{G} \cap W_{\text{Even}}$ and an Audrey attractor decomposition of $\mathcal{G} \cap W_{\text{Odd}}$.*

Ordered trees. Ordered trees are defined inductively; the trivial tree $\langle\rangle$ is an ordered tree and so is a sequence $\langle T_1, T_2, \ldots, T_\ell \rangle$, where T_i is an ordered tree for every $i = 1, 2, \ldots, \ell$. The trivial tree has only one node called the root, which is a leaf; and a tree of the form $\langle T_1, T_2, \ldots, T_\ell \rangle$ has the root with ℓ children, the root is not a leaf, and the i-th child of the root is the root of the ordered tree T_i.

Because the trivial tree $\langle\rangle$ has just one node, we sometimes write $\circ$ to denote it. If T is an ordered tree and i is a positive integer, then we use the expression $\left\langle T^i \right\rangle = \langle T, \ldots, T \rangle$ to denote the tree whose root has i children, each of which is the root of a copy of T. We also use the $\cdot$ symbol to denote concatenation of sequences, which in the context of ordered trees can be interpreted as sequential composition of trees by merging their roots; for example,

$$\left\langle \left\langle \circ^3 \right\rangle \right\rangle \cdot \left\langle \circ^4, \langle\langle\circ\rangle\rangle^2 \right\rangle = \left\langle \left\langle \circ^3 \right\rangle, \circ^4, \langle\langle\circ\rangle\rangle^2 \right\rangle = \langle \langle \circ, \circ, \circ \rangle, \circ, \circ, \circ, \circ, \langle\langle\circ\rangle\rangle, \langle\langle\circ\rangle\rangle \rangle .$$

For an ordered tree T, we write $\text{height}(T)$ for its *height* and $\text{leaves}(T)$ for the *number of its leaves*, which are defined inductively: the trivial tree $\langle\rangle = \circ$ has 1 leaf and its height is 1; the number of leaves of tree $\langle T_1, T_2, \ldots, T_\ell \rangle$ is the sum of the numbers of leaves of trees T_1, T_2, …, T_ℓ; and its height is 1 plus the maximum height of trees T_1, T_2, …, T_ℓ. Intuitively, an ordered tree is *equitable* if all its branches have the same height. We say that an ordered tree $T = \langle T_1, T_2, \ldots, T_\ell \rangle$ is

equitable if each T_i is equitable and they all have the same height. We say that an ordered tree is (n, h)-*small* if it has at most n leaves and its height is at most h.

Example 1. *The tree* $\left\langle \circ, \left\langle \circ^3 \right\rangle, \circ \right\rangle$ *is the tree* $\langle \circ, \langle \circ, \circ, \circ \rangle, \circ \rangle$ *it has* 5 *leaves and height* 3. *Similarly, the tree* $\left\langle \left\langle \circ^3 \right\rangle, \circ^4, \langle \langle \circ \rangle \rangle^2 \right\rangle$ *has* 9 *leaves and height* 4

Trees of Attractor Decompositions. The definition of an attractor decomposition is inductive and we define an ordered tree that reflects the hierarchical structure of an attractor decomposition. If d is even and

$$\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell) \rangle$$

is a Steven d-attractor decomposition then we define the *tree of attractor decomposition* $\mathcal{A}$, denoted by $T_{\mathcal{A}}$, to be the trivial ordered tree $\langle\rangle$ if $\ell = 0$, and otherwise, to be the ordered tree $\left\langle T_{\mathcal{A}_1}, T_{\mathcal{A}_2}, \ldots, T_{\mathcal{A}_\ell} \right\rangle$, where for every $i = 1, 2, \ldots, \ell$, tree $T_{\mathcal{A}_i}$ is the tree of attractor decomposition $\mathcal{A}_i$. Trees of Audrey attractor decompositions are defined analogously.

Observe that the sets $S_1, S_2, \ldots, S_\ell$ in an attractor decomposition as above are non-empty and pairwise disjoint, which implies that trees of attractor decompositions are small relative to the number of vertices and the number of distinct priorities in a parity game. The following proposition can be proved by routine structural induction.

Proposition 2.0.4. *If* $\mathcal{A}$ *is an attractor decomposition of an* (n, d)*-small parity game then its tree* $T_{\mathcal{A}}$ *is* $(n, \lceil d/2 \rceil + 1)$*-small.*

Labelled ordered tree. We define *labelled ordered trees* similar to ordered trees: the trivial tree $\langle\rangle$ is an $\mathbb{L}$-*labelled ordered tree* and so is the following sequence

$$\langle (a_1, \mathcal{L}_1), (a_2, \mathcal{L}_2), \ldots, (a_k, \mathcal{L}_k) \rangle,$$

where $\mathcal{L}_1$, $\mathcal{L}_2$, …, $\mathcal{L}_k$ are $\mathbb{L}$-labelled ordered trees, and a_1, a_2, …, a_k are distinct elements of a linearly ordered set $(\mathbb{L}, \leqslant)$ and $a_1 < a_2 < \cdots < a_k$ in that linear order. Note that ordered trees can be naturally viewed as $\mathbb{N}$-labelled ordered trees in which the sequence $a_1, a_2, \ldots, a_k$ is always an initial segment of the positive integers. We define the *unlabelling* of a labelled ordered tree

$$\langle (a_1, \mathcal{L}_1), (a_2, \mathcal{L}_2), \ldots, (a_k, \mathcal{L}_k) \rangle,$$

recursively, to be the ordered tree $\langle T_1, T_2, \ldots, T_k \rangle$, where T_i is the unlabelling of $\mathcal{L}_i$ for every $i = 1, 2, \ldots, k$. An $\mathbb{L}$-*labelling* of an ordered tree T is an $\mathbb{L}$-labelled tree $\mathcal{L}$ whose unlabelling is T. We define the *natural labelling* of an ordered tree $T = \langle T_1, \ldots, T_k \rangle$, again by a straightforward induction, to be the $\mathbb{N}$-labelled tree $\langle (1, \mathcal{L}_1), \ldots, (k, \mathcal{L}_k) \rangle$, where $\mathcal{L}_1, \ldots, \mathcal{L}_k$ are the natural labellings of trees T_1, $\ldots$, T_k. Unsurprisingly, the unlabelling of the natural labelling of an ordered tree $\mathcal{T}$ is tree $\mathcal{T}$ itself.

For an $\mathbb{L}$-labelled tree $\langle (a_1, \mathcal{L}_1), \ldots, (a_k, \mathcal{L}_k) \rangle$, its set of *nodes* is defined inductively to consist of the root $\langle\rangle$ and all the sequences in $\mathbb{L}^*$ of the form $\langle a_i \rangle \cdot v$, where $v \in \mathbb{L}^*$ is a node in $\mathcal{L}_i$ for some $i = 1, \ldots, k$, and where the symbol $\cdot$ denotes concatenation of sequences. For example, the natural labelling of tree $\left\langle \left\langle \circ^3 \right\rangle, \circ^4, \langle\langle \circ \rangle\rangle^2 \right\rangle$ has the set of nodes that consists of the following set of leaves $\langle 1, 1 \rangle$, $\langle 1, 2 \rangle$, $\langle 1, 3 \rangle$, $\langle 2 \rangle$, $\langle 3 \rangle$, $\langle 4 \rangle$, $\langle 5 \rangle$, $\langle 6, 1, 1 \rangle$, $\langle 7, 1, 1 \rangle$, and all of their prefixes. Indeed, the set of nodes of a labelled ordered tree is always prefix-closed. Moreover, if $L \subseteq \mathbb{L}^*$ then its closure under prefixes uniquely identifies a labelled ordered tree that we call the labelled ordered tree *generated* by L, and its unlabelling is the ordered tree generated by L. For example, the set $\{ \langle 1 \rangle, \langle 3, 1 \rangle, \langle 3, 4, 1 \rangle, \langle 6, 1 \rangle \}$ generates ordered tree $\langle \circ, \langle \circ, \langle \circ \rangle \rangle, \langle \circ \rangle \rangle$.

A node is the *ancestor* of another node if the former is a prefix of the latter. A node is a *descendant* of its ancestor. The node $\langle 3 \rangle$ is an ancestor of $\langle 3, 4, 1 \rangle$ and they are both descendants of the root $\langle\rangle$. A labelled ordered tree is a totally ordered set if we use the lexicographic order on the node and enforce that the ancestor is at most as large as the node itself. Using such an order, we say that a *parent* of a node is the largest ancestor of a node other than itself. We say a node is the *child* of its parent node. Since the root $\langle\rangle$ is the smallest node, it does not have a parent and the leaves of the tree do not have a child. Finally, two nodes are said to be *siblings* if they share the same parent.

Embedding ordered trees. Intuitively, an ordered tree T *can be embedded in* T' if T can be obtained from T' by pruning some subtrees. More formally, the trivial tree $\langle\rangle$ can be embedded in every ordered tree, and $\langle T_1, T_2, \ldots, T_k \rangle$ can be embedded in $\left\langle T'_1, T'_2, \ldots, T'_\ell \right\rangle$ if there are indices $i_1, i_2, \ldots, i_k$ such that $1 \leqslant i_1 < i_2 < \cdots < i_k \leqslant \ell$ and for every $j = 1, 2, \ldots, k$, we have that T_j can be embedded in T'_{i_j}.

Universal ordered trees. An ordered tree is (n, h)-*universal* [CDF+19] if every (n, h)-small ordered tree can be embedded in it. An obvious candidate for a universal tree is obtained by concatenating *all* (n, h)-small ordered trees in some order. But

an improvement to this naive universal tree is the complete n-ary tree of height h that can be defined by induction on h: if $h = 0$ then $C_{n,0}$ is the trivial tree $\langle\rangle$, and if $h > 0$ then $C_{n,h}$ is the ordered tree $\langle C^n_{n,h-1}\rangle$. The tree $C_{n,h}$ is obviously (n,h)-universal but its size is exponential in h.

Jurdziński and Lazić gave a construction of an (n,h)-universal tree $S_{n,h}$, whose size is at most $n^{\binom{\lceil \lg n\rceil + h + 1}{h}}$. An algorithm whose running time is a polynomial of the size of the tree would be both quasi-polynomial and also fixed parameter tractable with respect to h. To acknowledge their fundamental contribution, we refer to these trees as Jurdziński-Lazić universal trees.

For $g \geqslant 0$, let I_g be the trivial tree, that is the tree with exactly one leaf, of height g. For example, $I_1 = \langle\rangle$ and $I_3 = \langle\langle\langle\rangle\rangle\rangle = \langle\langle\circ\rangle\rangle$. Such an $S_{n,h}$ is constructed inductively as follows: $S_{1,h} = I_h$ and $S_{n,1} = \langle \circ^n \rangle$. Finally, the (n,h)-universal tree is declared to have two copies of the $(n/2,h)$-universal side attached—one to each side of the $(n,h-1)$-universal tree with an extra root node. More formally,

$$S_{n,h} = S_{n/2,h} \cdot \langle S_{n,h-1} \rangle \cdot S_{n/2,h}$$

Parys's (n,h)-universal tree, so named after the recursive attractor-based algorithm of Parys [Par19] is as follows: if $h = 0$ then $P_{n,h}$ is again defined to be the trivial tree $\langle\rangle$. If $h > 0$ then $P_{n,h}$ is defined to be the ordered tree

$$\left\langle P_{\lfloor n/2\rfloor,h-1}{}^{\lfloor n/2\rfloor} \right\rangle \cdot \langle P_{n,h-1}\rangle \cdot \left\langle \left(P_{\lfloor n/2\rfloor,h-1}\right)^{\lfloor n/2\rfloor} \right\rangle.$$

Ordering on bitstrings. We introduce the total linear order used by Jurdziński and Lazic on the set $\mathbb{W} = \{0,1\}^*$ of bit strings: for each bit $b \in \{0,1\}$, and for all bit strings $\beta, \beta' \in \{0,1\}^*$, if ε is the empty string, then we have

$$0\beta < \varepsilon < 1\beta, \qquad \text{and} \qquad b\beta < b\beta' \text{ iff } \beta < \beta'.$$

Theorem 2.0.5 ([JL17])**.** *The prefix-closure of the set of h-length sequences, where the word formed by the concatenation of sequences that have at most $\lceil \lg n \rceil$ bits, forms the set of nodes of a $\mathbb{W}$-labelled Jurdziński-Lazić universal tree $S_{n,h}$.*

Jurdziński and Morvan [JM20, JMT22] produced an algorithm that unified both McNaughton-Zielonka algorithm and its recent quasi-polynomial variants due to Parys [Par19], and due to Lehtinen, Schewe, and Wojtczak [LPSW22] by producing a unifying algorithm whose recursive subcalls were dictated by an interleaving of universal trees. In Algorithm 1, we reproduce the Jurdziński-Morvan algorithm to solve parity games, by stating procedure JM-Even (which they call $\text{Univ}_{\text{Even}}$)

and only remark that procedure JM-Odd is defined analogously. Their algorithm computes attractors in a similar manner to the original McNaughton-Zielonka algorithm, but with its recursive calls guided by two trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$ instead. They

Algorithm 1 The Jurdziński-Morvan Algorithm

Input: A game $\mathcal{G}$ with maximum (even) priority d, two trees $\mathcal{T}^{\text{Even}}$, and $\mathcal{T}^{\text{Odd}}$ whose heights are both $d/2 + 1$.

Output: A subset of vertices of the game $\triangleright$ These vertices are winning for Steven if $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$ are both universal.

1: **procedure** JM-Even$(\mathcal{G}, d, \mathcal{T}^{\text{Even}}, \mathcal{T}^{\text{Odd}})$
2: let $\mathcal{T}^{\text{Odd}} = \left\langle \mathcal{T}_1^{\text{Odd}}, \mathcal{T}_2^{\text{Odd}}, \ldots, \mathcal{T}_k^{\text{Odd}} \right\rangle$
3: $\mathcal{G}_1 \leftarrow \mathcal{G}$
4: **for** $i \leftarrow 1 \ldots k$ **do**
5: $D_i \leftarrow \pi^{-1}(d) \cap \mathcal{G}_i$
6: $A_i \leftarrow$ Steven attractor to D_i in $\mathcal{G}_i$
7: $\mathcal{G}_i' \leftarrow \mathcal{G}_i \setminus A_i$
8: $U_i \leftarrow$ JM-Odd$\left(\mathcal{G}_i', d-1, \mathcal{T}^{\text{Even}}, \mathcal{T}_i^{\text{Odd}}\right)$
9: $A_i' \leftarrow$ Audrey attractor to U_i in $\mathcal{G}_i$
10: $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus A_i'$
11: **end for**
12: **return** $V(\mathcal{G}_{k+1})$
13: **end procedure**

further remarked that on setting tree $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$ to the complete trees $C_{n,d/2+1}$ and $C_{n,d/2}$ respectively, Algorithm 1 simulates the recursive calls of McNaughton and Zielonka. If instead, these trees are declared to be Parys's universal trees $P_{n,d/2+1}$ and $P_{n,d/2}$, we get Parys's [Par19, LPSW22] recursive attractor-based algorithm. Finally, Lehtinen, Schewe and Wojtczak's [LSW19, LPSW22] improvement of Parys's algorithm can be obtained by setting the trees to the Jurdziński-Lazić universal trees $S_{n,d/2+1}$ and $S_{n,d/2}$.

In this thesis, especially in Part II, we present algorithms in which universal trees are implicitly present. These trees are never given as part of the input to any of the algorithms. We instead assume that our algorithm has access to the nodes of these trees and can perform elementary operations on them, such as finding the next sibling and finding the parent, efficiently. All the above-mentioned universal trees, and the ones we introduce in the future chapters, satisfy this requirement.

Rabin games. *Rabin games* are also played on a directed graph [Rab69, EJ91] between Steven and Audrey. A Rabin game consists of a directed graph (V, E) whose set of vertices V are partitioned among the two players, a start vertex and

two subsets of the set of colours $\{c_0\} \cup C$ for each vertex v denoted by G_v of a set *good* colours and a set R_v of *bad* colours. We require above that c_0 is a colour that is distinct from all colours in C and G_v does not contain c_0 for any vertex v. Such games are also called a (c_0, C)-colourful Rabin game.

An infinite path in the underlying graph satisfies the Rabin condition if there exists some colour such that among the set of vertices visited infinitely often, there is a vertex v that contains c in G_v, and none of the vertices v that appear infinitely often contain c in R_v.

Steven's positional strategy σ is defined similarly to his positional strategy in a parity game. A (positional) *Steven strategy* is a set $\sigma \subseteq E$ of edges such that:

- for every v that belongs to Steven, there is an edge $(v, u) \in \sigma$,
- for every v that belongs to Audrey, if $(v, u) \in E$ then $(v, u) \in \sigma$.

A positional strategy σ is winning for Steven if all infinite paths in the game restricted to the game graph (V, σ) satisfy the Rabin condition. Steven wins the Rabin game if he has some positional strategy to win in $\mathcal{G}$. Although in our introductory definition, we did not exclude the possibility of non-positional strategies, it is enough to consider only positional strategies. Rabin games are won by Steven using positional strategies, although Audrey might require winning strategies that are not positional to win a Rabin game [EJ88, EJ99]. Steven traps, Steven attractors, Audrey traps, and Audrey attractors are defined exactly as for parity games.

Observe that an (n, d)-small parity game on an graph (V, E) can be encoded as a (c_0, C)-colourful Rabin game on the same directed graph where $C = \{1, 2, \ldots, \lfloor d/2 \rfloor\}$ has $\lfloor d/2 \rfloor$ colours. We assign the good sets and bad sets to be

$$G_v = \{i \mid \pi(v) \geqslant 2i\} \qquad \text{and} \qquad B_v = \{i \mid \pi(v) \geqslant 2i + 1\},$$

for all $i \in \{1, \ldots, \lfloor d/2 \rfloor\}$ and all $v \in V$. It is routine to verify that any infinite path on such a (c_0, C)-colourful game is winning for Steven if and only if the corresponding path in the parity game is even.

Because the subsets of vertices for each colour form a chain with respect to the partial order induced by set-inclusion, the parity condition is also known as Rabin-chain condition.

We informally mention other related games that we refer to in this thesis.

Muller games. A Muller game [McN93] $\mathcal{G}$ consists of an arena that is a directed graph (V, E) whose set of vertices is partitioned among Steven and Audrey, a finite

subset of *tags* S, a tagging function that labels each vertex from V with a tag from S, and an objective $\mathcal{F}$ that is specified by a family of subsets of tags $\mathcal{F} \subseteq \mathcal{P}(S)$. We define (non-positional) strategies, plays, and plays respecting strategies similarly to parity games. A play is said to be winning for Steven if the set of tags among the set of vertices visited infinitely often by this play is in $\mathcal{F}$. Steven wins the game if he has a strategy, such that all plays that respect this strategy are winning for him.

Part I

Chapter 3

Solving nested fixpoint equations

Computing fixpoints is fundamental in computer science. The problem of identifying the solution to a system of nested fixpoint equations (NFEs) over finite lattices is known to be computationally equivalent to identifying the winner of a parity game [LBC+94, Sei96, BKMMP19]. However, most of the reductions involve an exponential increase in the size of the resulting parity game. The satisfiability problem of the coalgebraic μ-calculus has also been reduced to it [HS19]. Notably, breakthrough result of Calude et al. could be interpreted as an algorithm that solves specific kinds of fixpoint equations in quasi-polynomial time. Following their progress, there were several algorithms that were aimed at solving more general fix-point equations using universal trees [ANP21] and universal graphs [HS21]. Hausmann and Schröder [HS21] gave a quasi-polynomial algorithm to solve NFEs using progress measures on universal graphs. In parallel, Arnold, Niwiński, and Parys [ANP21] solved NFEs using the key result on decompositions of dominions similar to that of Jurdziński and Morvan's [JM20, JMT22] universal algorithm.

Within this context, our contribution is to provide a distinct perspective to solving nested fixpoints. We achieve this by transforming a nested fixpoint equation into a fixpoint game, inspired by the approach of Hausmann and Schröder [HS21], yet utilising Jurdziński and Morvan's universal algorithm parameterised by two trees as in [ANP21]. Note that fixpoint games are exponentially larger than the representation of a fixpoint equation. However, we provide a modification to the Jurdziński-Morvan algorithm, thereby ensuring that this modified algorithm operates on a specific type of subgames of these fixpoint games, termed "flowery subgames." By capitalising on the properties of flowery subgames, we bypass the exponential

representation typical for fixpoint games.

Comparatively, our approach, published in 2022 [JMT22] aligns with the underlying concepts of Arnold, Niwiński, and Parys' algorithm, as we also utilise a pair of trees to guide the computation process. However, we differ in two key aspects. Firstly, it should be noted that Arnold, Niwiński, and Parys [ANP21] also provide an asymmetrical version of their algorithm—using a technique of Seidl [Sei96]—whose running time is almost a square-root in the worst case than our proposed algorithm. Their asymmetric algorithm uses a technique of Seidl [Sei96], which ensures that the required number of function evaluations to solve the nested fixpoint equation is linear in the size of one universal tree, as opposed to ours, which requires time that is a square of the size of a universal trees. Secondly, our algorithm includes the computation of attractors—inherited from the Jurdziński-Morvan algorithm for solving parity games—which is absent in both the symmetric and asymmetric versions of the algorithms by Arnold, Niwiński, and Parys.

3.1 Nested fixpoint equations

Nested fixpoint equations, as the name suggests, are a system of equations, where each equation is a fixpoint equation defined on arbitrary functions over a finite lattice. In this chapter, however, we only consider nested fixpoint equations over finite powerset lattices.

Consider a finite set of elements U and its powerset lattice $\mathcal{P}(U)$. Let f be a monotone function (component-wise monotone) from $\mathcal{P}(U)^d$ to $\mathcal{P}(U)^d$. The function f can be expressed as a d-tuple $\langle f_1, \cdots, f_d \rangle$ of functions, where each function f_i, for $i \in \{1, \ldots, d\}$ is from $\mathcal{P}(U)^d$ to $\mathcal{P}(U)$ and each f_i is just the projection of the function f to the i-th component. Since there is a natural bijection from d-tuples of subsets of U to subsets of $(U \times [d])$, we instead denote f as a function from $\mathcal{P}(U \times [d])$ to $\mathcal{P}(U \times [d])$.

A nested fixpoint equation is a system of d fixpoint equations of the form:

$$X_i =_{\eta_i} f_i(X_1, \ldots, X_d) \qquad (*)$$

for i ranging over $1, \ldots d$, where $\eta_i = \nu$, refers to the greatest fixpoint operator, if i is odd, and $\eta_i = \mu$, which refers to the least fixpoint operator. We call a system such as Eq. $(*)$ a nested fixpoint equation and write $X =_\eta f(X)$ to depict it. One could consider a more general form of fixpoint equations where $\eta_i \in \{\mu, \nu\}$, but for simplicity of presentation, we restrict ourselves to the above.

The *solution* of a system of d equations as defined by Eq. $(*)$, is a subset of

$U \times [d]$, defined recursively as follows. The solution of the empty set of equations is the empty tuple. For a system of one or more fixpoint equations, we define a function f^{d-1} from subsets of U to subsets of $(U \times [d-1])$. This function f^{d-1} takes as input Y_d, a subset of U, and uses this input to fix $X_d = Y_d$ in the system of equations. The solution to the system of $d-1$ equations is obtained by fixing X_d to be Y_d as the output of f^{d-1}. We finally say the solution of the system of equations is $\left(f^{d-1}(Y_d), Y_d\right)$, where $Y_d = \eta_d \left(\lambda X_d . f_d(f^{d-1}(X_d), X_d)\right)$.

Fixpoint Games. For a system f of nested fixpoint equations, we define an *equivalent* parity game $\mathcal{G}_f$, called a *fixpoint game*. The solution of the parity game $\mathcal{G}_f$ correlates to the solution of the system of nested fixpoint equation defined by $X =_\eta f(X)$ [BKMMP19, HS21].

The underlying graph in the game $\mathcal{G}_f$ so defined is (V_f, E_f) where the set of vertices V_f consists of the set and the powerset of $U \times [d]$, more precisely, it is exactly the set $(U \times [d]) \cup \{v_A \mid A \subseteq U \times [d]\}$. The vertices corresponding to elements of the set $(U \times [d])$ belong to Steven and the ones corresponding to subsets of the same set belong to Audrey. The priority function of $\mathcal{G}_f$, represented by π_f, assigns Steven's vertices (u, i) to i, whereas all vertices belonging to Audrey are assigned priority 0. The edges from a vertex (u, i), belonging to Steven in $\mathcal{G}_f$, lead to the set of Audrey's vertices $\{v_A \mid (u, i) \in f(A)\}$ and the edges from a vertex v_A, belonging to Audrey, lead to the set of Steven's vertices $\{(u, i) \mid (u, i) \in A\}$.

Theorem 3.1.1 ([BKMMP19], Theorem 4.8)**.** *For a system consisting of d fixpoint equations $X_i =_\eta f_i(X_1, \ldots, X_d)$, whose solution is $\langle Y_1, \ldots, Y_i \rangle$, the element $u \in Y_i$ if and only if Steven wins from the vertex (u, i) in the corresponding fixpoint game $\mathcal{G}_f$.*

Due to Theorem 3.1.1 of Baldan et al., [BKMMP19], we henceforth deal only with fixpoint games in order to solve our fixpoint equations.

3.2 Solving fixpoint games

We provide a way to solve fixpoint games by modifying the Jurdziński-Morvan algorithm. We define a specific kind of subgames that we call *flowery* subgames and show that they are pertinent for solving fixpoint games using attractor decomposition algorithms.

Intuitively, for two non-empty subsets of the set $(U \times [d])$, the flowery subgame $\mathcal{S}(X, Y)$ represents a subgame of $\mathcal{G}_f$ whose set of vertices consists of

- all vertices of Steven belonging to Y, resembling the core of a flower;
- all of Audrey's vertices v_A where A is a subset of X intersecting non-trivially with Y, resembling the petal of a flower.

Definition 3.2.1 (Flowery subgames)**.** *For a nested fixpoint game $\mathcal{G}_f$, obtained from a system of d equations over the lattice $\mathcal{P}(U)$, and two subsets $\varnothing \subsetneq Y \subseteq X \subseteq U \times [d]$, we define the flowery subgame $\mathcal{S}(X, Y)$ to be the subgame induced by the vertices $Y \uplus \{v_A \mid A \subseteq X \text{ and } A \cap Y \neq \varnothing\}$.*

We also call a subgame $\mathcal{G}'$ flowery if there are subsets X and Y of $(U \times [d])$ such that this subgame $\mathcal{G}'$ is equal to $\mathcal{S}(X, Y)$.

In the game $\mathcal{G}_f$, on removing vertices that have no outgoing edges along with the respective attractors to these sets of vertices, that is, Audrey attractors to Steven's vertices with no outgoing edges and vice versa, we get a flowery subgame. Moreover, the following lemma reassures us that all significant operations performed by the universal algorithm for parity games on flowery subgames, result in flowery subgames.

Lemma 3.2.2 (Flowery subgame lemma)**.** *The subgames $\mathcal{G}_i$ computed by the (modified) procedure* JM-Even *in the Jurdziński-Morvan algorithm (as in Algorithm 1) are flowery if the input game $\mathcal{G}$ is a flowery subgame. In particular, $\mathcal{G}_{k+1}$, which is the subgame returned by the procedure, is flowery.*

An analogous lemma holds for Audrey's procedure JM-Odd. The attractor to a set of vertices during a run of the algorithm can be computed by at most $d|U|$ many evaluations of f on subsets of $U \times [d]$. Therefore, we can solve nested fixpoint games in quasi-polynomial time using an attractor decomposition algorithm. This approach resembles Algorithm 1, with only minor adaptations required. Specifically, the subgames are replaced by flowery subgames, represented by two subsets of U. Furthermore, the computation of attractors is replaced with suitably defined modifications tailored for flowery subgames, as outlined in the statement of Lemma 3.2.5.

Theorem 3.2.3. *The modified universal algorithm that solves nested fixpoint equations on trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$ makes $|\mathcal{T}^{\text{Odd}}| \cdot |\mathcal{T}^{\text{Even}}|$ many recursive calls. Each recursive call makes at most $2d|U|$ function evaluations of f.*

Plugging in quasi-polynomial universal trees, we get the following theorem as a corollary.

Theorem A. *The modified Jurdziński-Morvan algorithm that computes nested fixpoint equations takes quasi-polynomial time and polynomial space, when the underlying trees are quasi-polynomial sized universal trees.*

We dedicate the remainder of the chapter to proving Lemma 3.2.2 and Theorem A. Whenever we want to denote the fixpoint obtained by repeated application of a monotone function f on a set, we call this f^*. Before we embark on the proofs, we would like to call attention to the following property of flowery subgames. It shows how complements of two specific kinds of flowery sets result in another flowery subgame. We will use this property in several of our proofs.

Property 1. *For $A \subseteq Y \subseteq X \subseteq (U \times [d])$, we have:*

$$\mathcal{S}(X, Y) \setminus \mathcal{S}(X, A) = \mathcal{S}(X \setminus A, Y \setminus A).$$

Notice that $\mathcal{S}(X \setminus A, Y \setminus A) = \mathcal{S}(Z \cup W, W)$, where $Z = X \setminus Y$ and $W = Y \setminus A$.

Consider the following proposition useful in the proof of the Lemma 3.2.2.

Proposition 3.2.4. *Given a fixpoint game $\mathcal{G}_f$, after removing the Steven attractor to the set of Audrey's vertices with no outgoing edges and the Audrey attractor to Steven's vertices with no outgoing edges, we are left with a flowery subgame.*

Proof. The game $\mathcal{G}_f$ contains exactly the vertices in the subgame $\mathcal{S}(U \times [d], U \times [d])$ along with $v_\varnothing$.

- Initially, we remove the only Audrey vertex with no outgoing edge: $v_\varnothing$, along with its Steven attractor. The Steven attractor to $v_\varnothing$ in $\mathcal{G}_f$ is exactly all the vertices of the flowery set $\mathcal{S}(C, C)$ and $v_\varnothing$, where $C = f^*(\varnothing)$ is winning for Steven. The remaining subgame after removing these vertices is the flowery subgame $\mathcal{S}(U \times [d], (U \times [d]) \setminus C)$ from Property 1.

- Let us call the flowery subgame obtained from the above procedure $\mathcal{S}(X, Y)$. Observe that if $Y \subseteq f(X)$, then there is always an outgoing edge for each vertex in the subgame. If not, we remove the Audrey attractor to the set of Steven's vertices with no outgoing edges: $Y \cap \overline{f(X)}$. The complement of this Audrey attractor turns out to be the flowery subgame $\mathcal{S}(X, Y \setminus \overline{f(X)})$ from Property 1.

□

Assuming now that we always have outgoing edges in flowery subgames, we consider the following lemma, which shows how we can compute attractors to sets in these subgames with at most $d \cdot |U|$ many calls to the function f.

Let us now prove Lemma 3.2.2. We proceed by proving a stronger statement than the one in Lemma 3.2.8. To prove Lemma 3.2.8, in turn, we need Lemma 3.2.5. Lemma 3.2.5 intuitively asserts that attractors of specific flowery subgames are specific flowery subgames whose complement is also flowery.

Lemma 3.2.5. *In a flowery subgame $\mathcal{G} = \mathcal{S}(X, Y)$:*

(a) the Steven attractor to a set of Steven's vertices $A \subseteq Y$ in $\mathcal{G} = \mathcal{S}(X, Y)$ where $Z = X \setminus Y$ is

$$\mathcal{S}(Z \cup \mathrm{Pre}^*_{\mathcal{G},\mathrm{Even}}(A), \mathrm{Pre}^*_{\mathcal{G},\mathrm{Even}}(A))$$

where $\mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A) = (f(Z \cup A) \cap Y) \cup A$;

(b) the Audrey attractor to a set of Steven's vertices A or a subgame $\mathcal{S}(X, A)$ in $\mathcal{G} = \mathcal{S}(X, Y)$ is

$$\mathcal{S}(X, \mathrm{Pre}^*_{\mathcal{G},\mathrm{Odd}}(A))$$

where $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) = (\overline{f(X \setminus A)} \cap Y) \cup A$.

We will break down our Lemma into Propositions 3.2.6 and 3.2.7, which will result in Corollary 3.2.7.1, from which Lemma 3.2.5 follows.

Proposition 3.2.6. *In a flowery subgame $\mathcal{G} = \mathcal{S}(X, Y)$ and $A \subseteq Y$, the vertices of the flowery subgame $\mathcal{S}(Z \cup \mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A), \mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A))$ are exactly those from which Steven has a strategy to visit A in at most three steps, where $\mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A) = (f(Z \cup A) \cap Y) \cup A$.*

Proof. We will argue about vertices from which Steven has a strategy to visit vertices in A in at most one, two and three steps below.

(1) Consider any Audrey vertex v_B where $B \subseteq Z \cup A$ and the intersection of B with A is non-empty. From such a v_B, in one step, Steven can ensure that a play reaches A. All such vertices v_B along with the core A are exactly denoted by the vertices of the subgame $\mathcal{S}(Z \cup A, A)$.

(2) We will show that, from any of Steven's vertex $(u, i) \in \mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A) = f(Z \cup A) \cap Y\} \cup A$, there is a strategy for him to reach a vertex in A owned by him in at most two steps. We will show that

($\Rightarrow$) in one step, Steven can move to some Audrey vertex $v_B \in \mathcal{S}(Z \cup A, A)$;

($\Leftarrow$) from vertices not in $\mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A)$, all of Steven's outgoing edges lead to a vertex not in $\mathcal{S}(Z \cup A, A)$.

To show the forward direction, let $(u,i) \in (f(Z \cup A) \cap Y) \cup A$. If $(u,i) \in A$ then we are done, if not, the strategy for Steven from (u,i) is to choose the Audrey vertex $v_{Z \cup A}$. Such an edge exists since $(u,i) \in f(Z \cup A)$, and this Audrey vertex is in the flowery subgame $\mathcal{S}(Z \cup A, A)$.

To show the reverse direction,consider $(u,i) \notin f(Z \cup A) \cup A$ but $(u,i) \in Y$. All outgoing edges from the Steven vertex (u,i) lead to an Audrey vertex v_B in $\mathcal{S}(X,Y)$ such that B has some element other than from Z or A, that is, $B \setminus (Z \cup A) \neq \varnothing$. This follows from the monotonicity of f along with our assumption that $(u,i) \notin f(Z \cup A)$. After one step, the game is at an Audrey vertex v_B that it is not in $\mathcal{S}(Z \cup A, A)$.

(3) The argument to conclude that $\mathcal{S}(X, \mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A))$ is exactly the set we desire is similar to (1). □

Proposition 3.2.7. *In a flowery subgame $\mathcal{G} = \mathcal{S}(X,Y)$ and for a set of Steven vertices $A \subseteq Y$ in it, Audrey has a strategy to visit some vertex from the set of Steven vertices A in at most three steps from all the vertices in the flowery subgame $\mathcal{S}(X, \mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A))$, where $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) = (\overline{f(X \setminus A)} \cap Y) \cup A$. This subgame is also contains all vertices from which Audrey has such a strategy.*

Proof. We show the set of vertices from which Steven has a strategy to visit vertices in A in at most one, two and three steps below.

(1) From vertex v_B where B of X which intersects with A non-trivially, Audrey would be able to reach a vertex in A in at most one step. This is exactly all Audrey's vertices in the flowery subgame $\mathcal{S}(X, A)$.

(2) We will show that, in one step, Audrey has a strategy to visit the subgame $\mathcal{S}(X, A)$ from the vertices in $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) \cup A$. We do this by showing inclusion in two directions.

($\Rightarrow$) Consider $(v,j) \in \mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) = (\overline{f(X \setminus A)} \cap Y) \cup A$. If $(v,j) \notin A$, then $(v,j) \in Y$ and $\overline{f(X \setminus A)}$. Mainly note that $(v,j) \notin f(X \setminus A)$. Since all subgames are such that there is always an outgoing edge and given that f is monotone, any Audrey vertex v_B in $\mathcal{S}(X,Y)$ which has an edge to it from (v,j) must be such that $B \cap A \neq \varnothing$. For any choice successors

from (v, j) of Steven will lead to a vertex B that intersects with A and hence there is a strategy for Audrey to move to a vertex in (u, i) in the set $B \cap A$.

($\Leftarrow$) Now we need to show a strategy for Steven to avoid $\mathcal{S}(X, \mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A))$, for two steps from all other Steven vertices. Let us denote $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A)$ by W. Note that the complement of $\mathcal{S}(X, W)$ in $\mathcal{S}(X, Y)$ is $\mathcal{S}(X \setminus W, Y \setminus W)$. Also notice that

$$Y \setminus W = Y \setminus \left(\overline{f(X \setminus A)} \cup A\right).$$

So, any $(w, j) \in Y \setminus Z$ is in Y and since $(w, j) \notin W$, $(w, j) \in f(X \setminus A)$. This means that, from any such (w, j), Steven can choose the vertex v_B in $\mathcal{S}(X \setminus W, Y \setminus W)$ where $B \subseteq X \setminus A$, making sure that in the next step Audrey will not be able to take the play to Steven's vertex in A.

(3) From the structure of the game, it is easy to see that any v_B such that B intersects with $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) \cup A$ would be able to visit an element in $\mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A) \cup A$, which we have shown is exactly the set of vertices from which Audrey could force the play in at most two steps to visit A. □

From the proof of the Propositions 3.2.6 and 3.2.7, we can extend these to show the corollary below from which Lemma 3.2.5 follows.

Corollary 3.2.7.1. *In a flowery subgame $\mathcal{G} = \mathcal{S}(X, Y)$ and $A \subseteq Y$,*

- *The flowery subgame $\mathcal{S}(Z \cup \mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A), \mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A))$ is the set of vertices from which Steven has a strategy to visit the vertices in $\mathcal{S}(Z \cup A, A)$ in at most two steps, where $\mathrm{Pre}_{\mathcal{G},\mathrm{Even}}(A) = (f(Z \cup A) \cap Y) \cup A$;*
- *The vertices of $\mathcal{S}(X, \mathrm{Pre}_{\mathcal{G},\mathrm{Odd}}(A))$ is the set of vertices from which Audrey has a strategy to visit a vertex in $\mathcal{S}(X, A)$ in at most two steps.*

Lemma 3.2.5 follows naturally from Corollary 3.2.7.1 thus concluding the proof of Lemma 3.2.5.

We will now proceed to the main proof of the section, where we prove a (slightly) stronger version of our flowery subgame lemma below.

Lemma 3.2.8. *(i) If* JM-Even *is run on a flowery subgame $\mathcal{S}(X, Y)$, then in all iterations in the for-loop, the subgame $\mathcal{G}_i = \mathcal{S}(X \setminus A'_i, Y \setminus A'_i)$, where $A'_i \subseteq Y$.*

(ii) If JM-Odd *is run on a flowery subgame $\mathcal{S}(X, Y)$, then for each i, the subgame $\mathcal{G}_i = \mathcal{S}(X, Y \setminus A'_i)$, where $A'_i \subseteq Y$.*

Proof. We will prove this by induction on the sum of the number of vertices in these subgames and the number of vertices on which these calls are made. For the base case, with an empty set irrespective of any priority, the above statement is trivially true. We will now prove that (i) and (ii) hold for games with at least one vertex and trees $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$. The proof follows from Lemma 3.2.5 and by an induction on the size of the game and tree.

(i) Since $\mathcal{G}_1 = \mathcal{G} = \mathcal{S}(X, Y)$, we show that if $\mathcal{G}_i$ is of the form $\mathcal{S}(X \setminus A'_i, Y \setminus A'_i)$ where $A_i \subseteq Y$, then $\mathcal{G}_{i+1}$ is also of the form $\mathcal{S}(X \setminus A'_{i+1}, Y \setminus A'_{i+1})$, where $A_{i+1} \supseteq A_i$. For convenience, we will call $X \setminus A'_i$ as X_i and $Y \setminus A'_i$ as Y_i. We will show that $\mathcal{G}_{i+1}$ is of the form $\mathcal{S}(X \setminus A'_{i+1}, Y \setminus A'_{i+1})$ by showing that in fact it is $\mathcal{S}(X_i \setminus A'_{i+1}, Y_i \setminus A'_{i+1})$ for some $A'_{i+1} \subseteq Y$. First, note that $\mathcal{G}'_i$ is a subgame of obtained by removing A_i, which is the Steven attractor to the set D_i containing only Steven vertices which, moreover, have the highest even priority vertices in $\mathcal{G}_i$. From Lemma 3.2.6, we have for $Z = X \setminus Y$,

$$\mathcal{G}_i \setminus A_i = \mathcal{S}(X_i, Y'_i) \setminus \mathcal{S}(Z \cup \text{Pre}^*_{\mathcal{G}_i,\text{Even}}(D_i), \text{Pre}^*_{\mathcal{G}_i,\text{Even}}(D_i))$$

Since $Z = X \setminus Y = X_i \setminus Y_i$, we have

$$\mathcal{G}'_i = \mathcal{G}_i \setminus A_i = \mathcal{S}(X_i, Y_i) \setminus \text{Pre}^*_{\mathcal{G}_i,\text{Even}}(D_i)$$

The U_i computed by performing JM-Odd on $\mathcal{G}'_i$ must be of the form $\mathcal{S}(X_i, Z_i)$ for $Z_i \subseteq Y_i$ by induction and the attractor to U_i, must be of the form $\mathcal{S}(X_i, W_i)$ from Proposition 3.2.6. Hence,

$$\mathcal{G}_{i+1} = \mathcal{S}(X_i, Y_i) \setminus \mathcal{S}(X_i, W_i) = \mathcal{S}(X_i \setminus W_i, Y_i \setminus W_i).$$

(ii) We will show that if $\mathcal{G}_i$ is of the form $\mathcal{S}(X, Y_i)$, then $\mathcal{G}_{i+1}$ is of the form $\mathcal{S}(X, Y_{i+1})$ for $Y_{i+1} \subseteq Y_i$. In each iteration i, the Audrey attractor to D_i in $\mathcal{G}_i$ is of the form $\mathcal{S}(X, A_i)$. This shows that $\mathcal{G}'_i$, which is obtained by removing the Audrey attractor $\mathcal{S}(X, A_i)$ from $\mathcal{G}_i$ is of the form $\mathcal{S}(X_i \setminus A_i, Y_i \setminus A_i)$. The procedure JM-Odd on $\mathcal{G}_i$ gives U_i of the form $\mathcal{S}(X_i \setminus W_i, Y_i \setminus W_i)$ by induction, and Steven attractor to the set $\mathcal{S}(X_i \setminus W_i, Y_i \setminus W_i)$ would be of the flowery subgame $\mathcal{S}(X_i \setminus W'_i, Y_i \setminus W'_i)$ for some $W'_i \subseteq W_i$. So, $\mathcal{G}_{i+1}$, which is obtained from removing this Steven attractor from $\mathcal{G}_i$ would be obtained as follows

$$\mathcal{G}_{i+1} = \mathcal{G}_i \setminus \mathcal{S}(X_i \setminus W'_i, Y_i \setminus W'_i) = \mathcal{S}(X, Y_i \setminus W'_i). \qquad \square$$

3.3 Concurrent parity games

We consider a concurrent stochastic version of parity games in this subsection [Sha53, McN93, dAH00], to illustrate a corollary of Theorem A to the field of parity games research. We consider the two player version as studied by Chatterjee, Alfredo and Henzinger in [CAH11]. For an exact definition of these concepts, we refer the reader again to the work of Chatterjee, Alfaro, and Henzinger [CAH11] and restrict ourselves to an informal discussion here. These games are played between Steven and Audrey, but instead of partitioning the vertices among the two players, they take simultaneous actions at each vertex and the token moves to a neighbour depending on the actions of both players. The outcome is depicted in the form of a matrix, where the columns represent Steven's choice of actions, and the rows correspond to Audrey's choice. When both players at each turn pick a row and a column simultaneously, their outcome is determined by the entry of this matrix, where each entry of this matrix is just another vertex of this game.

One might also consider a stochastic version, where the outcome of simultaneous actions is based on a pre-decided probability distribution. For the stochastic version, each entry is a further probability distribution among the vertices. A token is moved from a start vertex, based on the simultaneous choice that determines the next vertex, thus creating an infinite play. This infinite play is then required to satisfy the parity condition, that is, the highest priority seen must be even, for Steven to win this play. Unlike the vanilla version of parity games, both players are allowed to use a randomised strategy, i.e., a strategy where the next action is proposed with the help of a probability distribution. A vertex is called *limit-winning* for Steven (respectively Audrey) if for all ϵ, Steven has a (randomised) strategy such that all plays from that vertex obeying this strategy with probability at least 1. That is, for all $0 < \epsilon \leqslant 1$, there is a strategy that ensures with probability at least $(1 - \epsilon)$ that the path that obeys the strategy is winning for Steven. Note that turn-based parity games can also be encoded as a concurrent parity game, where from Steven's vertices, Audrey's actions do not affect where the token moves to the next and vice-versa.

The decision question at hand is to determine whether a vertex is a limit-winning for Steven. In concurrent parity games, unlike our definition of parity games, a player might need both infinite memory and randomisation to win these games. We refer the reader to the work of Chatterjee, Alfaro, and Henzinger [CAH11] for a rigorous definition of the above games along with examples for the claims above. In their paper, they show that solving concurrent parity games is in $\mathsf{NP} \cap \mathsf{coNP}$ as

a corollary of the following theorem.

Theorem 3.3.1 ([CAH11, Theorem 5, Lemma 29 and Lemma 30])**.** *Limit-winning in a concurrent parity game can be expressed as a system of nested fixpoint equations over the powerset lattice of the set of edges with alternation depth at most* $2d$ *for a function, whose evaluation involves solving another system of nested fixpoint equations also with depth at most* $2d$*.*

An easy corollary from Theorem A along with Theorem 3.3.1, we have the following.

Corollary 3.3.1.1. *Limit-winning vertices of Steven and Audrey in concurrent parity games can be determined in quasi-polynomial time.*

Chapter 4

The Strahler number of a parity game

Strahler Number. The Strahler number of a rooted tree is the largest height of a perfect binary tree that is its minor. This concept was initially introduced by Horton (1945) and later formalised by Strahler (1952). They developed this notion during their morphological exploration of river networks within the field of hydrogeology. The Strahler number's applicability isn't limited to hydrogeology—it has found relevance across various scientific disciplines, including botany, anatomy, neurophysiology, physics, and molecular biology, all of which deal with branching patterns. In the realm of computer science, Ershov [Ers58] recognised the Strahler number as the minimum number of registers necessary to evaluate an arithmetic expression. Subsequently, this concept has experienced numerous resurgences across different domains within computer science. Notable surveys by Knuth [Knu73], Viennot [Vie90], and Esparza, Luttenberger, and Schlund [ELS16] have captured its reappearance and importance within the computer science landscape.

Lehtinen's algorithm for solving parity games. The major breakthrough in the quest for a polynomial-time algorithm for parity games was achieved by Calude, Jain, Khoussainov, Li, and Stephan [CJK$^+$17], who gave the first quasi-polynomial algorithm. Other quasi-polynomial algorithms have been developed soon after by Jurdziński and Lazić [JL17], and Lehtinen [LB20], and by Fearnley et al. [FJdK$^+$19].

Lehtinen's algorithm provides a new parameter for parity games: the register number. She further argued that games of bounded register number can be solved in polynomial time. It was known that if the underlying graphs have bounded tree-

width [Obd03, FL11, Gan15, Sta23], clique-width [Obd07], DAG-width [BDHK06], Kelly-width [HK07] and entanglement [BG04], the same holds. We make two remarks here, the first is that solving parity games is not in FPT for any of the above parameters and the second remark—also made by Lehtinen—is that none of the other parameters on the underlying graph co-incide with the register number of parity games.

Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić, and Parys [CDF$^+$19] introduced the concepts of *universal trees* and *separating automata*, and argued that all the aforementioned quasi-polynomial algorithms were intimately linked to them. Czerwiński et. al's lower bound argument on Lehtinen's algorithm varies from the others. This can be attributed to Lehtinen's algorithm producing a non-deterministic parity automaton as a separating automaton as opposed to a deterministic safety automaton of the other algorithms. The lowerbound is obtained indirectly by arguing that the safety automaton that can be derived from a non-determinisitc parity automaton with some good-for-separation properties has the lower bound induced by universal trees.

Parys [Par20] has tried to reconcile this difference as well as offered some running-time improvements to Lehtinen's algorithm, but it remains significantly worse than the bounds of Jurdziński and Lazić [JL17], as well as its Fearnley, Jain, de Keijzer, Schewe, Stephan, and Wojtczak [FJdK$^+$19], and the improvements proposed by Dell'Erba and Schewe [DS22] as Lehtinen's algorithm always requires at least quasi-polynomial working space. Moreover, Parys [Par20]'s proposed method to improve the runtime of Lehtinen's algorithm re-defines the register game introduced by Lehtinen and restricts the strategies of one player to "positional" strategies.

Our Contributions. This chapter contains work published by Daviaud and Jurdziński and the author in 2020 [DJT20]. Here, we propose the Strahler number as a parameter that measures the structural complexity of dominia in a parity game and that governs the computational complexity of the most efficient algorithms currently known for solving parity games. We establish that the Strahler number is a robust, and hence natural, parameter by proving that it coincides with its version based on trees of progress measures and with the register number defined by Lehtinen [Leh18, LB20].

In this chapter, we give a natural characterisation of Lehtinen's register number in terms of attractor decompositions and its trees. We recall verbatim the definition of an attractor decomposition here for ease of reference.

Attractor Decompositions. If $\mathcal{G}$ is a parity game in which all priorities do not exceed a non-negative even number d then we say that

$$\mathcal{A} \;=\; \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$$

is a *Steven d-attractor decomposition* [DJL18, DJL19] of $\mathcal{G}$ if:

- A is the Steven attractor to the (possibly empty) set of vertices of priority d in $\mathcal{G}$;

and setting $\mathcal{G}_1 = \mathcal{G} \setminus A$, for all $i = 1, 2, \ldots, \ell$, we have:

- S_i is a non-empty trap for Audrey in $\mathcal{G}_i$ in which every vertex priority is at most $d - 2$;
- $\mathcal{A}_i$ is a Steven $(d-2)$-attractor decomposition of subgame $\mathcal{G} \cap S_i$;
- A_i is the Steven attractor to S_i in $\mathcal{G}_i$;
- $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$;

and the game $\mathcal{G}_{\ell+1}$ is empty. If $d = 0$ then we require that $\ell = 0$.

Recall that if d is even and

$$\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$$

is a Steven d-attractor decomposition then we define the *tree of attractor decomposition* $\mathcal{A}$, denoted by $T_{\mathcal{A}}$, to be the trivial ordered tree $\langle\rangle$ if $\ell = 0$, and otherwise, to be the ordered tree $\left\langle T_{\mathcal{A}_1}, T_{\mathcal{A}_2}, \ldots, T_{\mathcal{A}_\ell}\right\rangle$, where for every $i = 1, 2, \ldots, \ell$, tree $T_{\mathcal{A}_i}$ is the tree of attractor decomposition $\mathcal{A}_i$.

Strahler Numbers of an Ordered Tree. The *Strahler number* $\mathrm{Str}(T)$ of a tree T is defined to be the largest height of a perfect binary tree that is a minor of T. Alternatively, it can be defined by the following structural induction: the Strahler number of the trivial tree $\langle\rangle = \circ$ is 1; and if $T = \langle T_1, \ldots, T_\ell\rangle$ and s is the largest Strahler number of trees $T_1, \ldots, T_\ell$, then $\mathrm{Str}(T) = s$ if there is a unique i such that $\mathrm{Str}(T_i) = s$, and $\mathrm{Str}(T) = s + 1$ otherwise. Recall that we denote the trivial tree with one node $\langle\rangle$ by $\circ$. For example, we have

$$\mathrm{Str}\left(\left\langle\left\langle \circ^3\right\rangle, \circ^4, \langle\langle\circ\rangle\rangle^2\right\rangle\right) = 2$$

because $\mathrm{Str}(\circ) = \mathrm{Str}(\langle\langle\circ\rangle\rangle) = 1$ and $\mathrm{Str}\left(\left\langle \circ^3\right\rangle\right) = 2$.

Proposition 4.0.1. *For every (n, h)-small tree T, we have* $\mathrm{Str}(T) \leqslant h$ *and* $\mathrm{Str}(T) \leqslant \lfloor \lg n \rfloor + 1$.

Strahler number of a parity game. We define the *Strahler number of an attractor decomposition* $\mathcal{A}$, denoted by $\mathrm{Str}(\mathcal{A})$, to be the Strahler number $\mathrm{Str}(T_{\mathcal{A}})$ of its tree $T_{\mathcal{A}}$. We define the *Strahler number of a parity game* to be the maximum of the smallest Strahler numbers of attractor decompositions of the largest Steven and Audrey dominions, respectively.

4.1 Strahler number bounds register number

In this section, we establish a connection between the register number of a parity game defined by Lehtinen [Leh18] and the Strahler number thus defined. Moreover, we argue that from every Steven attractor decomposition of Strahler number k, we can derive a dominion strategy for Steven in the k-register game. Once we establish the Strahler number upper bound on the register number, we are faced with the following two natural questions:

Question 4.1.1. *Do the Strahler and the register numbers coincide?*

Question 4.1.2. *Can the relationship between Strahler and register numbers be exploited algorithmically, in particular, to improve the running time and space complexity of solving register games studied by Lehtinen [Leh18] and Parys [Par20]?*

In Chapters 4 and 5 we answer them both positively (Lemma 4.1.3 and Theorem B, and Theorem C, respectively).

For every positive number k, a Steven *k-register game* on a parity game $\mathcal{G}$ is another parity game $\mathcal{R}^k(\mathcal{G})$ whose vertices, edges, and priorities will be referred to as *states*, *moves*, and *ranks*, respectively, for disambiguation. The states of the Steven k-register game on $\mathcal{G}$ are either pairs $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle)$ or triples $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle, p)$, where v is a vertex in $\mathcal{G}$, $d \geqslant r_k \geqslant r_{k-1} \geqslant \cdots \geqslant r_1 \geqslant 0$, and $1 \leqslant p \leqslant 2k + 1$. The former states have rank 1 and the latter have rank p. Each number r_i, for $i = k, k-1, \ldots, 1$, is referred to as the value of the i-th register in the state. Steven owns all states $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle)$ and the owner of vertex v in $\mathcal{G}$ is the owner of states $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle, p)$ for every p. How the game is played by Steven and Audrey is determined by the available moves:

- at every state $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle)$, Steven picks i, such that $0 \leqslant i \leqslant k$, and *resets* registers $i, i-1, i-2, \ldots, 1$, leading to state $\left(v, \left\langle r'_k, \ldots, r'_{i+1}, r'_i, 0, \ldots, 0 \right\rangle, p\right)$

of rank p and with updated register values, where:

$$p = \begin{cases} 2i & \text{if } i \geqslant 1 \text{ and } \max(r_i, \pi(v)) \text{ is even,} \\ 2i+1 & \text{if } i = 0, \text{ or if } i \geqslant 1 \text{ and } \max(r_i, \pi(v)) \text{ is odd;} \end{cases}$$

$r'_j = \max\big(r_j, \pi(v)\big)$ for $j \geqslant i+1$, and $r'_i = \pi(v)$;

- at every state $(v, \langle r_k, r_{k-1}, \ldots, r_1 \rangle, p)$, the owner of vertex v in $\mathcal{G}$ picks an edge (v, u) in $\mathcal{G}$, leading to state $(u, \langle r_k, r_{k-1}, \ldots, r_1 \rangle)$ of rank 1 and with unchanged register values.

For example, at state $(v, \langle 9, 6, 4, 4, 3 \rangle)$ of rank 1, if the priority $\pi(v)$ of vertex v is 5 and Steven picks $i = 3$, this leads to state $(v, \langle 9, 6, 5, 0, 0 \rangle, 7)$ of rank $2i + 1 = 7$ because $\max(r_3, \pi(v)) = \max(4, 5) = 5$ is odd, $r'_4 = \max(r_4, \pi(v)) = \max(6, 5) = 6$, and $r'_3 = \pi(v) = 5$.

Observe that the first components of states on every cycle in game $\mathcal{R}^k(\mathcal{G})$ form a (not necessarily simple) cycle in parity game $\mathcal{G}$; we call it the cycle in $\mathcal{G}$ *induced* by the cycle in $\mathcal{R}^k(\mathcal{G})$. If a cycle in $\mathcal{R}^k(\mathcal{G})$ is even (that is, the highest state rank on it is even) then the induced cycle in $\mathcal{G}$ is also even. Lehtinen [Leh18, Lemmas 3.3 and 3.4] has shown that a vertex v is in the largest Steven dominion in $\mathcal{G}$ if and only if there is a positive integer k such that a state $(v, \overline{r})$, for some register values $\overline{r}$ is in the largest Steven dominion in $\mathcal{R}^k(\mathcal{G})$. Lehtinen and Boker [LB20, a comment after Definition 3.1] have further clarified that for every k, if a player has a dominion strategy in $\mathcal{R}^k(\mathcal{G})$ from a state whose first component is a vertex v in $\mathcal{G}$, then they also have a dominion strategy in $\mathcal{R}^k(\mathcal{G})$ from every state whose first component is v. This allows us to say without loss of rigour that a vertex v in $\mathcal{G}$ is in a dominion in $\mathcal{R}^k(\mathcal{G})$.

By defining the *(Steven) register number* [Leh18, Definition 3.5] of a parity game $\mathcal{G}$ to be the smallest number k such that all vertices v in the largest Steven dominion in $\mathcal{G}$ are in a Steven dominion in $\mathcal{R}^k(\mathcal{G})$, and by proving the $1 + \lg n$ upper bound on the register number of every (n, d)-small parity game [Leh18, Theorem 4.7], Lehtinen has contributed a novel quasi-polynomial algorithm for solving parity games, adding to those by Calude et al. [CJK$^+$17] and Jurdziński and Lazić [JL17].

Lehtinen [Leh18, Definition 4.8] has also considered the concept of a Steven *defensive dominion strategy* in a k-register game (for brevity, we call it a k-defensive strategy): it is a Steven dominion strategy on a set of states in $\mathcal{R}^k(\mathcal{G})$ in which there is no state of rank $2k + 1$. Alternatively, the same concept can be formalised by

defining the *defensive k-register game* $\mathcal{D}^k(\mathcal{G})$, which is played exactly like the k-register game $\mathcal{R}^k(\mathcal{G})$, but in which Audrey can also win just by reaching a state of rank $2k + 1$. Note that the game $\mathcal{D}^k(\mathcal{G})$ can be thought of as having the winning criterion for Steven as being a conjunction of a parity and a safety criteria, and the winning criterion for Audrey as a disjunction of a parity and a reachability criteria. Routine arguements allow to extend positional determinacy from parity games to such games with combinations of parity, and safety or reachability winning criteria.

We follow Lehtinen [Leh18, Definition 4.9] by defining the *(Steven) defensive register number* of a Steven dominion D in $\mathcal{G}$ as the smallest number k such that Steven has a defensive dominion strategy in $\mathcal{R}^k(\mathcal{G})$ on a set of states that includes all $(v, \langle r_k, \ldots, r_1 \rangle)$ for $v \in D$, and such that r_k is an even number at least as large as every vertex priority in D. We propose to call it the *Lehtinen number* of a Steven dominion in $\mathcal{G}$ to honour Lehtinen's insight that led to this concept. We also define the Lehtinen number of a vertex in $\mathcal{G}$ to be the smallest Lehtinen number of a Steven dominion in $\mathcal{G}$ that includes the vertex, and the Lehtinen number of a parity game as the Lehtinen number of its largest Steven dominion. We also note that the register and the Lehtinen numbers of a parity game nearly coincide (they differ by at most one), and hence the conclusions of our analysis of the latter also apply to the former.

Lemma 4.1.3. *The Lehtinen number of a parity game is no larger than its Strahler number.*

The arguments used in our proof of this lemma are similar to those used in the proof of the main result of Lehtinen [Leh18, Theorem 4.7]. Similar results also appears in the author's Master's thesis [The19], which shows that the Strahler number of a progress measure tree bounds the Lehtinen number. However, we provide a similar statement using attractor decompositions instead. Our contribution here is to pinpoint the Strahler number of an attractor decomposition as the structural parameter of a dominion that naturally bounds the number of registers used in Lehtinen's construction of a defensive dominion strategy.

Proof of Lemma 4.1.3. Consider a parity game $\mathcal{G}$ and let d be the least even integer no smaller than any of the priority in $\mathcal{G}$. Consider a Steven d-attractor decomposition $\mathcal{A}$ of $\mathcal{G}$ of Strahler number k. We construct a defensive k-register strategy for Steven on $\mathcal{R}^k(\mathcal{G})$. The strategy is defined inductively on the height of $\mathcal{T}_{\mathcal{A}}$, and has the additional property of being $\mathcal{G}$*-positional* in the following sense: if $\left((v, \langle r_k, \ldots, r_1 \rangle), \left(v, \left\langle r'_k, \ldots, r'_1 \right\rangle, p\right)\right)$ is a move then the register reset by Steven only depends on v, not on the values in the registers. Similarly, if the move

$((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ is such that v is owned by Steven, u only depends on v and not on the values of the registers or p.

Strategy for Steven. If $\mathcal{A} = \langle A \rangle$, then $\mathcal{G}$ is exactly the set of vertices of priority d and of its Steven attractor. In this case, Steven follows the strategy induced by the reachability strategy in A to the set of vertices of priority d, only resetting register r_1 immediately after visiting a state with first component a vertex of priority d in $\mathcal{G}$. More precisely, the Steven defensive strategy is defined with the following moves:

- $((v, \langle r_1 \rangle), (v, \langle r_1 \rangle, 1))$ if v is not a vertex of priority d in $\mathcal{G}$;
- $\left((v, \langle r_1 \rangle), \left(v, \left\langle r_1' \right\rangle, 2\right)\right)$ if v is a vertex of priority d in $\mathcal{G}$ and $r_1' = \max(r_1, d)$ is even;
- $\left((v, \langle r_1 \rangle), \left(v, \left\langle r_1' \right\rangle, 3\right)\right)$ if v is a vertex of priority d in $\mathcal{G}$ and $r_1' = \max(r_1, d)$ is odd (we state this case for completeness but this will never occur);
- $((v, \langle r_1 \rangle, p), (u, \langle r_1 \rangle))$ where (v, u) belongs to the Steven reachability strategy from A to the set of vertices of priority d in $\mathcal{G}$.

Note that this strategy is $\mathcal{G}$-positional.

Suppose now that $\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell) \rangle$ and that it has Strahler number k. For all $i = 1, 2, \ldots, \ell$, let k_i be the Strahler number of $\mathcal{A}_i$. By induction, for all i, we have a Steven defensive k_i-register strategy σ_i, which is $(\mathcal{G} \cap S_i)$-positional, on a set of states Ω_i in $\mathcal{R}^{k_i}(\mathcal{G} \cap S_i)$ including all the states $\left(v, \left\langle r_{k_i}, \ldots, r_1 \right\rangle\right)$ for $v \in S_i$ and r_{k_i} an even number at least as large as every vertex priority in S_i. Let Γ_i be the set of states in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ defined as all the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ for $v \in S_i$ if $k_i \neq k$ and as the union of the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ for $v \in S_i$ and Ω_i, otherwise.

The strategy σ_i induces a strategy on Γ_i in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ by simply ignoring registers $r_{k_i+1}, \ldots, r_k$, and using $(\mathcal{G} \cap S_i)$-positionality to define moves from the states not in Ω_i. More precisely, in a state $(v, \langle r_k, \ldots, r_1 \rangle)$, Steven resets register j if and only if register j is reset in a state $\left(v, \left\langle r_{k_i}', \ldots, r_1' \right\rangle\right)$ of Ω_i according to σ_i. This is well defined by $(\mathcal{G} \cap S_i)$-positionality. Similarly, we add moves $((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ to the strategy if and only if there is a move $\left(\left(v, \left\langle r_{k_i}', \ldots, r_1' \right\rangle, p'\right), \left(u, \left\langle r_{k_i}', \ldots, r_1' \right\rangle\right)\right)$ in σ_i. This is again well-defined by $(\mathcal{G} \cap S_i)$-positionality.

This strategy is denoted by τ_i. Note that τ_i is a defensive k-register strategy on Γ_i, which is $\mathcal{G}$-positional.

The Steven defensive strategy in $\mathcal{R}^k(\mathcal{G})$ is defined by the following moves, where S denotes the set of vertices of priority d in $\mathcal{G}$:

- On the set of states with first component a vertex of S_i, the moves are given by τ_i.
- On the set of states with first component a vertex of $A_i \setminus S_i$, Steven uses the strategy induced by the reachability strategy from A_i to S_i, without resetting any registers.
- On $\mathcal{R}^k(\mathcal{G} \cap (A \setminus S))$, Steven uses the strategy induced by the reachability strategy from A to S, without resetting any registers.
- On the set of states with first component a vertex of S,
  - $((v, \langle r_k, \ldots, r_1 \rangle), (v, \langle d, 0, \ldots, 0 \rangle, p))$ where v is a vertex in S and $p = 2k$ if $\max(r_k, d)$ is even and $p = 2k + 1$ otherwise.
  - $((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ for some uniquely chosen u such that (v, u) in E if v is owned by Steven and for all u such that (v, u) in E if v is owned by Audrey.

Observe that this strategy is $\mathcal{G}$-positional.

Correctness of the Strategy. We prove now that the strategy defined above is indeed a defensive k-register strategy. We proceed by induction on the height of $\mathcal{T}_\mathcal{A}$ and define a set of states Γ, including all the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ such that v is a vertex of $\mathcal{G}$.

Base Case: If the height of $\mathcal{T}_\mathcal{A}$ is 1 and $\mathcal{A} = \langle A \rangle$, let Γ be the set of states $(v, \langle r_1 \rangle)$ and $(v, \langle r_1 \rangle, p)$ with v a vertex of $\mathcal{G}$, $1 \leqslant r_1 \leqslant d$ and p being either 1 or 2. It is easy to see that the strategy defined above is a defensive dominion strategy on this set.

Inductive step: If $\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell) \rangle$ with Strahler number k and k_i being the Strahler number of $\mathcal{A}_i$ for all i (note that $k_i \leqslant k$ for all i, and by definition of Strahler number, there is at most one m such that $k_m = k$), we define Γ to be the set comprising the union of the Γ_i and all the states of the form $(v, \langle r_k, \ldots, r_1 \rangle)$ and $(v, \langle r_k, \ldots, r_1 \rangle, p)$ with v a vertex of $(A_i \setminus S_i) \cup A$ and $1 \leqslant p \leqslant 2k$.

Case 1: For each i, $k_i < k$.

We first show that Γ is a trap for Audrey for the strategy defined above, showing that rank $2k + 1$ can never be reached (implying that the strategy is defensive). This comes from the fact that the register of rank k is only reset in a state

$(v, \langle r_k, \ldots, r_1 \rangle)$ with v in S. Since $\max(r_k, d) = d$ is even then this leads to a state $(v, \langle d, 0, \ldots, 0 \rangle, 2k)$. Otherwise, register k is never reset, so a state with rank $2k+1$ cannot be reached.

Consider now any cycle in $\mathcal{R}^k(\mathcal{G})$ with moves restricted to the strategy constructed above. If this cycle contains a state whose first component is a vertex of S, then as explained above, the highest rank in the cycle is $2k$. Otherwise, the cycle is necessarily in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ for some i. By induction, τ_i is winning and so the cycle is even.

Case 2: There is a unique m such that $k_m = k$.

We first show that a state of rank $2k + 1$ is never reached. Observe that register k is reset in two places: (1) immediately after a state with first component a vertex of S is visited, (2) if register k is reset by τ_m. In the first case, similarly as shown above, a state of rank $2k$ is reached. In the second case, register k is either reset in a state $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$, and similarly as above, a state of rank $2k$ is reached, or in a state of Ω_i. In this case, as τ_i is defensive on Ω_i by induction, a state of rank $2k + 1$ cannot be reached, and the highest rank that can be reached is $2k$.

Proving that every cycle is even is similar to the previous case. □

4.2 Strahler number is bounded by register number

In this section we prove that every parity game whose Lehtinen number is k has an attractor decomposition of Strahler number at most k. In other words, we establish the Lehtinen number upper bound on the Strahler number, which together with Lemma 4.1.3 provides a positive answer to Question 4.1.1 in the theorem below.

Theorem B. *The Strahler number of a parity game is equal to its Lehtinen number.*

When talking about strategies in parity games, we only considered positional strategies, for which it was sufficient to verify the parity criterion on (simple) cycles. Instead, we explicitly consider the parity criterion on infinite paths here, which we find more convenient to establish properties of Audrey strategies in the proof of Theorem B.

First, we introduce the concepts of *tight* and *offensively optimal* attractor decompositions.

Definition 4.2.1. *A Steven d-attractor decomposition $\mathcal{A}$ of $\mathcal{G}$ is* tight *if Audrey has a winning strategy from at least one state in $\mathcal{D}^{\mathrm{Str}(\mathcal{A})-1}(\mathcal{G})$ in which the value of register* $\mathrm{Str}(\mathcal{A}) - 1$ *is d.*

By definition, the existence of a tight Steven d-attractor decomposition on a parity game implies that the Lehtinen number of the game is at least its Strahler number, from which Theorem B follows. Offensive optimality of an attractor decomposition, the concept we define next, may seem less natural and more technical than tightness, but it facilitates our proof that every game has a tight attractor decomposition.

Definition 4.2.2. *Let $\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$ be a Steven d-attractor decomposition, let games $\mathcal{G}_i$ for $i = 1, 2, \ldots, \ell$ be as in the definition of an attractor decomposition, let A_i' be the Audrey attractor of the set of vertices of priority $d-1$ in $\mathcal{G}_i$, and let $\mathcal{G}_i' = \mathcal{G}_i \setminus A_i'$. We say that $\mathcal{A}$ is* offensively optimal *if for every $i = 1, 2, \ldots, \ell$, we have:*

- *Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}\big(\mathcal{G}_i'\big)$;*
- *Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)}\big(\mathcal{G}_i' \setminus S_i\big)$.*

Proving that every offensively optimal Steven attractor decomposition is tight (Lemma 4.2.5), and that every Steven dominion in a parity game has an offensively optimal Steven attractor decomposition (Lemma 4.2.6), will complete the proof of Theorem B. We first give two propositions that will be useful in the proofs.

Proposition 4.2.3. *For every parity game $\mathcal{G}$ and non negative integer k, if Audrey has a dominion strategy from every state of $\mathcal{D}^k(\mathcal{G})$ then Audrey has a dominion strategy on $\mathcal{R}^k(\mathcal{G})$.*

Proof. For every state s of $\mathcal{D}^k(\mathcal{G})$, Audrey has a winning strategy τ_s on $\mathcal{D}^k(\mathcal{G})$ starting in s. We construct a dominion strategy for her on $\mathcal{R}^k(\mathcal{G})$: after every visit to a state of rank $2k+1$, Audrey follows τ_s, where s is the first state that follows on the path and whose rank is smaller than $2k+1$. This defines a dominion strategy on $\mathcal{R}^k(\mathcal{G})$. □

Proposition 4.2.4. *If $\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$ is an offensively optimal Steven d-attractor decomposition, then for every $i = 1, 2, \ldots, \ell$, Audrey has a dominion strategy on $\mathcal{R}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$ (and also a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$).*

Proof. Let i in $\{1, 2, \ldots, \ell\}$. Consider the following strategy in $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$:

- On the set of states whose vertex components are in A_i', Audrey follows a strategy induced by the reachability strategy in A_i' to a vertex of priority $d-1$ (picking any move if v is of priority $d-1$);

- In states whose vertex component is in $\mathcal{G}_i'$, Audrey plays a $(k-1)$-register dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}\big(\mathcal{G}_i'\big)$. Such a strategy exists by the definition of offensive optimality.

This strategy is indeed an Audrey dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$, because any play either visits a state whose first component is a vertex in A_i' infinitely often, or it eventually remains in $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}\big(\mathcal{G}_i'\big)$. In the former case, the play visits a state whose first component is a vertex of priority $d-1$ infinitely often. In the latter case, the strategy is a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}\big(\mathcal{G}_i'\big)$.

Finally, we use Proposition 4.2.3 to turn this Audrey dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$ into an Audrey dominion strategy on $\mathcal{R}^{\mathrm{Str}(\mathcal{A}_i)-1}(\mathcal{G}_i)$. □

Lemma 4.2.5. *Every offensively optimal Steven attractor decomposition is tight.*

Proof. Let the attractor decomposition $\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$ be an offensively optimal d-attractor decomposition of a parity game and let $k = \mathrm{Str}(\mathcal{A})$. We construct a strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ that is winning for her from at least one state in which the value of register $k-1$ is d. We define $\mathcal{G}_i'$ and A_i' as in Definition 4.2.2.

Case 1: $\mathrm{Str}(\mathcal{A}_i) = k$ for some unique i in $\{1, \ldots, \ell\}$. In this case, we show that Audrey has a dominion strategy on $\mathcal{D}^{k-1}(\mathcal{G}_i)$. Since $\mathcal{G}_i$ is a trap for Steven in $\mathcal{G}$, this gives the desired result. This directly follows from Proposition 4.2.4.

Case 2: There are $1 \leqslant i < j \leqslant \ell$ such that $\mathrm{Str}(\mathcal{A}_i) = \mathrm{Str}\big(\mathcal{A}_j\big) = k-1$. We construct a strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ that is winning for her from all states in $\mathcal{G}_j$ whose register $k-1$ has value d. Firstly, since $\mathcal{A}$ is offensively optimal, Audrey has a dominion strategy on $\mathcal{D}^{k-1}\big(\mathcal{G}_i' \setminus S_i\big)$, denoted by τ_i, and a dominion strategy on $\mathcal{R}^{k-2}\big(\mathcal{G}_i'\big)$, denoted by τ_i'. Moreover, by Proposition 4.2.4, we have that Audrey has a dominion strategy, denoted by τ_j, on $\mathcal{R}^{k-2}\big(\mathcal{G}_j\big)$ (note that $\mathcal{G}_j$ is a trap for Steven in $\mathcal{G}$). Consider the following strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$, starting from a state whose vertex component is in $\mathcal{G}_j$ and register $k-1$ has value d:

- As long as the value of register $k-1$ is larger than $d-1$, Audrey follows the strategy induced by τ_j, while ignoring the value of register $k-1$.
- If the value in register $k-1$ is at most $d-1$:
  - In states whose vertex component is in A_i', Audrey follows a strategy induced by the reachability strategy from A_i' to a vertex of priority $d-1$ (picking any move if the vertex has priority $d-1$);

– In states whose vertex component is in $\mathcal{G}'_i \setminus S_i$ and whose register $k-2$ has value at most $d-2$, Audrey follows τ_i;
– In states whose vertex component is in $\mathcal{G}'_i$ and whose register $k-1$ has value $d-1$, Audrey follows the strategy induced by τ'_i, while ignoring the value of regiser $k-1$.

Audrey plays any move if none of the above applies.

We argue that this strategy is winning for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ from states whose vertex component is in $\mathcal{G}_j$ and register $k-1$ has value d. Consider an infinite path that starts in such a state. As long as register $k-1$ has value d, Audrey follows τ_j. If Steven never resets register $k-1$ then Audrey wins. Otherwise, once register $k-1$ has been reset, its value is at most $d-1$. Note that $\mathcal{G}_j$ is included in $A'_i \cup (\mathcal{G}'_i \setminus S_i)$. If register $k-1$ has a value smaller than $d-1$, and the play never visits a state whose vertex component is in A'_i, then Audrey has followed τ_i along the play (she has never left $\mathcal{G}'_i \setminus S_i$ as the only way for Steven to go out $\mathcal{G}'_i \setminus S_i$ is to go to A'_i) and wins. Otherwise, the play visits a state whose vertex component is in A'_i, and so it visits a state whose vertex component has priority $d-1$, leading to a state in which register $k-1$ has value $d-1$. Finally, if a state whose vertex component is in A'_i is visited infinitely many times then Audrey wins. Otherwise, Audrey eventually plays according to τ'_i. If Steven never resets register $k-1$ then Audrey wins. Otherwise, if Steven resets register $k-1$, which at this point has value $d-1$, a state of rank $2k-1$ is visited and Audrey wins. □

Lemma 4.2.6. *Every Steven dominion in a parity game has an offensively optimal Steven attractor decomposition.*

Proof. Consider a parity game $\mathcal{G}$ whose vertices form a Steven dominion. Let k be the Lehtinen number of $\mathcal{G}$ and let d be the largest even value such that $\pi^{-1}(\{d, d-1\}) \neq \varnothing$. We construct an offensively optimal Steven attractor decomposition by induction.

If $d = 0$, it is enough to consider $\langle A \rangle$, where A is the set of all vertices in $\mathcal{G}$.

If $d > 1$, let A be the Steven attractor of the set of vertices of priority d in $\mathcal{G}$. Let $\mathcal{G}_0 = \mathcal{G} \setminus A$. If $\mathcal{G}_0 = \varnothing$ then $\langle A \rangle$ is an offensively optimal Steven attractor decomposition for $\mathcal{G}$. Otherwise, $\mathcal{G}_0$ is a non-empty trap for Steven in $\mathcal{G}$ and therefore $\mathcal{G}_0$ has a Lehtinen number at most k. Let A' be the Audrey attractor of all the vertices of priority $d-1$ in the sub-game $\mathcal{G}_0$ and let $\mathcal{G}'_0 = \mathcal{G}_0 \setminus A'$.

Given a positive integer b, let L^b be the largest dominion in $\mathcal{G}'_0$ such that Steven has a dominion strategy on $\mathcal{D}^b\big(\mathcal{G}'_0\big)$. We define m to be the smallest number such that $L^m \neq \varnothing$ and let $S_0 = L^m$. We show that $m \leqslant k$. To prove this, we

construct an Audrey dominion strategy on $\mathcal{D}^b(\mathcal{G}_0)$ for all b such that $L^b = \varnothing$. Since the Lehtinen number of $\mathcal{G}_0$ is at most k, this implies that $m \leqslant k$. The Audrey dominion strategy on $\mathcal{D}^b(\mathcal{G}_0)$, assuming $L^b = \varnothing$, is as follows

- if the vertex component of a state is in A' then Audrey uses the strategy in A' induced by the reachability strategy to vertices of priority $d-1$;
- if the vertex component of a state is in $\mathcal{G}'_0$ then Audrey uses her dominion strategy on $\mathcal{D}^b\big(\mathcal{G}'_0\big)$, which exists because the Steven dominion L^b in $\mathcal{D}^b\big(\mathcal{G}'_0\big)$ is empty.

Any play following the strategy defined above and visiting infinitely often a state of $\mathcal{D}^b\big(\mathcal{G}_0 \cap A'\big)$ is winning for Audrey. A play following the above strategy and remaining eventually in $\mathcal{D}^b\big(\mathcal{G}'_0\big)$ is also winning for Audrey.

Let $\mathcal{A}_0$ be the $(d-2)$-attractor decomposition of S_0 obtained by induction. In particular, $\mathcal{A}_0$ is offensively optimal.

Let A_0 be the Steven attractor to S_0 in $\mathcal{G}_0$ and let $\mathcal{G}_1 = \mathcal{G}_0 \setminus A_0$. Subgame $\mathcal{G}_1$ is a trap for Steven and therefore it is a Steven dominion. Consider an offensively optimal Steven d-attractor decomposition $\mathcal{A}' = \langle \varnothing, (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$ of $\mathcal{G}_1$ obtained by induction.

We claim that $\mathcal{A} = \langle A, (S_0, \mathcal{A}_0, A_0), (S_1, \mathcal{A}_1, A_1), \ldots, (S_\ell, \mathcal{A}_\ell, A_\ell)\rangle$ is an offensively optimal Steven d-attractor decomposition of $\mathcal{G}$. Since $\mathcal{A}'$ is offensively optimal, it is enough to show that:

- Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)-1}\big(\mathcal{G}'_0\big)$,
- Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)}\big(\mathcal{G}'_0 \setminus S_0\big)$.

Since $\mathcal{A}_0$ is offensively optimal, Audrey has a winning strategy from at least one state in $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)-1}(S_0)$, by Lemma 4.2.5, and hence $m \geqslant \mathrm{Str}\,(\mathcal{A}_0)$.

So, by choice of m, Steven does not have a defensive dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)-1}\big(\mathcal{G}'_0\big)$ from any state. This means that Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)-1}\big(\mathcal{G}'_0\big)$.

Moreover, by construction of S_0, Audrey has a dominion strategy on the subgame $\mathcal{D}^m\big(\mathcal{G}'_0 \setminus S_0\big)$. This implies that Audrey has a dominion strategy on the subgame $\mathcal{D}^{\mathrm{Str}(\mathcal{A}_0)}\big(\mathcal{G}'_0 \setminus S_0\big)$. □

4.3 Strahler number of progress measures

Consider a parity game $\mathcal{G}$ in which all vertex priorities are at most an even number d.

If $(\mathbb{L}, \leqslant)$ is a well-founded linear order then we write sequences in $\mathbb{L}^{d/2}$ in the following form $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle$, and for every priority $p \in \{0, 1, \ldots, d\}$, we define the *p-truncation* of $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle$, denoted by $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle|_p$, to be the sequence $\langle m_{d-1}, \ldots, m_{p+2}, m_p \rangle$ if p is odd and $\langle m_{d-1}, \ldots, m_{p+3}, m_{p+1} \rangle$ if p is even. We use the lexicographic order $\leqslant_{\text{lex}}$ to linearly order the set $\mathbb{L}^* = \bigcup_{i=0}^{\infty} \mathbb{L}^i$.

A *Steven progress measure* [EJ91, Jur00, JL17] on a parity game $\mathcal{G}$ is a map $\mu : V \to \mathbb{L}^{d/2}$ such that for every vertex $v \in V$:

- if $v \in V_{\text{Even}}$ then there is a μ-progressive edge $(v, u) \in E$;
- if $v \in V_{\text{Odd}}$ then every edge $(v, u) \in E$ is μ-progressive;

where we say that an edge $(v, u) \in E$ is *μ-progressive* if:

- if $\pi(v)$ is even then $\mu(v)|_{\pi(v)} \geqslant_{\text{lex}} \mu(u)|_{\pi(v)}$;
- if $\pi(v)$ is odd then $\mu(v)|_{\pi(v)} >_{\text{lex}} \mu(u)|_{\pi(v)}$.

We define *the tree of a progress measure μ* to be the ordered tree generated by the image of V under μ.

Theorem 4.3.1 ([EJ91, Jur00, JL17])**.** *There is a Steven progress measure on a parity game $\mathcal{G}$ if and only if every vertex in $\mathcal{G}$ is in its largest Steven dominion. If game $\mathcal{G}$ is (n, d)-small then the tree of a progress measure on $\mathcal{G}$ is $(n, d/2+1)$-small.*

We define the *Steven progress-measure Strahler number* of a parity game $\mathcal{G}$ to be the smallest Strahler number of a tree of a progress measure on $\mathcal{G}$. The following theorem refines and strengthens Theorems 2.0.3 and 4.3.1 by establishing that the Steven Strahler number and the Steven progress-measure Strahler number of a parity game nearly coincide.

Theorem 4.3.2. *The Steven Strahler number and the Steven progress-measure Strahler number of a parity game differ by at most* 1*.*

The translations between progress measures and attractor decompositions that are used in the proof are as given by Daviaud, Jurdziński, and Lazić [DJL18]; here we point out that they do not increase the Strahler number of the underlying trees by more than 1.

Proof of Theorem 4.3.2. Let $\mathcal{G}$ be a (n, d)-small parity game. To prove Theorem 4.3.2 we will prove the following two lemmas.

Lemma 4.3.3. *If $\mathcal{G}$ is a parity game where all the vertices belong to Audrey and $\mathcal{G}$ has a Steven attractor decomposition of Strahler number k, then it has a Steven progress measure of Strahler number at most $k+1$.*

Proof. Let $\mathcal{G}$ be a parity game where all the vertices belong to Audrey. The proof is by induction on the height of the tree of a Steven attractor decomposition of $\mathcal{G}$.

Inductive statement. Given a d-attractor decomposition $\mathcal{A}$ of $\mathcal{G}$ and its tree $\mathcal{T}_{\mathcal{A}}$ of height h, there is a progress measure tree $\mathcal{T}$ of height h and an embedding f from $\mathcal{T}_{\mathcal{A}}$ to $\mathcal{T}$ such that all the nodes of $\mathcal{T}$ which are not in the image of f are leaves.

Base case. If the height of $\mathcal{T}$ is at most 1, then the d-attractor decomposition is $\langle A \rangle$. Let C be the set of vertices, which do not have priority d. Consider the topological order: $u < v$ if there is a path from v to u in A. We consider the tree $\left\langle \circ^{|C|} \right\rangle$ and μ, which maps the vertices of priority d to its root and the vertices in C to leaves, respecting the topological order, i.e. if $u < v$ then u is mapped to a node that is a larger sibling of the node v is mapped to. This defines a progress measure of Strahler number at most 2.

Induction step. Consider a Steven-d-attractor decomposition:

$$\mathcal{A} = \langle A, (S_1, \mathcal{A}_1, A_1), \ldots, (S_j, \mathcal{A}_j, A_j) \rangle .$$

Let $\mathcal{T}_{\mathcal{A}_i}$ be the tree of $\mathcal{A}_i$ and $\mathcal{G}_i$ as defined in the definition of an attractor decomposition.

Inductively, for all i, there is a progress measure tree $\mathcal{T}_i$ (and an associated progress measure mapping μ_i) of the same height as $\mathcal{T}_{\mathcal{A}_i}$ and an embedding f_i from $\mathcal{T}_{\mathcal{A}_i}$ to $\mathcal{T}_i$ such that all the nodes of $\mathcal{T}_i$ which are not in the image of f_i are leaves.

Let us construct a progress measure tree for $\mathcal{G}$ as follows. Let $C_i = A_i \setminus S_i$ for each i and C be the set of nodes in A that have priority at most $d-1$. Set:

$$\mathcal{T} = \left\langle \circ^{|C|}, \mathcal{T}_1, \circ^{|C_1|}, \ldots, \mathcal{T}_j, \circ^{|C_j|} \right\rangle .$$

Set μ to be a mapping from the set of vertices of $\mathcal{G}$ to the nodes of $\mathcal{T}$, which extends μ_i on vertices in S_i, maps the vertices of priority d to the root of the tree, the vertices in C to the first $|C|$ children of the root and the vertices in C_i to the corresponding $|C_i|$ children of the root which respects the topological ordering in $\mathcal{G}$ as viewed as a graph, i.e. if for vertices u and v in C, resp. C_i, there is a path from

u to v in C, resp. C_i, then u is mapped to a node that appears on the right of the node v is mapped to.

By construction and induction hypothesis, the tree $\mathcal{T}$ embeds $\mathcal{T}_\mathcal{A}$ and the only nodes that are not images of nodes in $\mathcal{T}_\mathcal{A}$ are leaves. Moreover, $\mathcal{T}$ is a progress measure tree with mapping μ by induction hypothesis and the construction, which is compatible with the Steven reachability strategy on A and the A_i's.

The lemma follows from the fact that the Strahler number of a tree increases by at most 1 when leaves are added to it. □

Lemma 4.3.4. *If $\mathcal{G}$ has a Steven progress measure of Strahler number k, then it has a Steven attractor decomposition of Strahler number at most k.*

Proof. We will prove the following by induction, which proves the lemma:

Inductive statement. Given an (n, d)-small parity game $\mathcal{G}$ where d is even and a progress measure tree $\mathcal{T}$ on $\mathcal{G}$, there exist a Steven attractor decomposition whose tree embeds in $\mathcal{T}$.

Remark 1. *Given a progress measure mapping μ on $\mathcal{G}$ and its corresponding progress measure tree $\mathcal{T}$, and given a trap R for Audrey in $\mathcal{G}$, the restriction of μ to the vertices in R is a progress measure with the tree induced by the nodes images of the vertices of R by μ.*

Base case. For games with one vertex, any progress measure tree on $\mathcal{G}$ and any tree of a Steven attractor decomposition are $\langle\rangle$. Therefore the induction hypothesis is satisfied.

Induction step. Let $\mathcal{G}$ be an (n, d)-small parity game where d is the least even integer no smaller than any priority in $\mathcal{G}$ and let $\mathcal{T}$ be a progress measure tree on $\mathcal{G}$.

Case 1: If the highest priority in $\mathcal{G}$ is even and equal to d. Let A be the Steven attractor of the set of vertices of priority d. Let $\mathcal{G}' = \mathcal{G} \setminus A$. As $\mathcal{G}'$ is a trap for Audrey in $\mathcal{G}$, the tree $\mathcal{T}'$ induced by the nodes images of the vertices in $\mathcal{G}'$ in $\mathcal{T}$ is a progress measure tree of $\mathcal{G}'$. By induction hypotheses, there exist a Steven attractor decomposition $\mathcal{A}$ of $\mathcal{G}'$ whose tree $\mathcal{T}_\mathcal{A}$ embeds in $\mathcal{T}'$. By appending A to $\mathcal{A}$, one gets a Steven attractor decomposition of $\mathcal{G}$ of same tree $\mathcal{T}_\mathcal{A}$, which then embeds in $\mathcal{T}$.

Case 2: If the highest priority in $\mathcal{G}$ is odd and equal to $d - 1$.

No vertex is mapped to the root in the progress measure tree $\mathcal{T}$. Let $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_j$ be the subtrees, children of the root of $\mathcal{T}$. Let us note that vertices of priority $d-1$ cannot be mapped to nodes in $\mathcal{T}_0$ as they would not have progressive outgoing edges if that was the case. Let S_0 be the set of vertices mapped to nodes in $\mathcal{T}_0$ and let A_0 be the Steven attractor of S_0 in $\mathcal{G}$. We can assume that S_0 is non-empty (otherwise we remove $\mathcal{T}_0$ from $\mathcal{T}$ and start again).

Let $\mathcal{G}' = \mathcal{G} \setminus A_0$. As $\mathcal{G}'$ is a subgame, trap for Audrey, the tree $\mathcal{T}'$ with subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_j$ is a progress measure tree on $\mathcal{G}'$. By induction, one gets a Steven attractor decomposition:

$$\mathcal{A}' = \langle \varnothing, (S_1, \mathcal{A}_1, A_1), \ldots, (S_j, \mathcal{A}_j, A_j) \rangle$$

whose tree embeds in $\mathcal{T}'$.

Now, let us prove that S_0 is a trap for Audrey. Let u be in S_0 and v be one of its successor. For (u, v) to be progressive, v has to be mapped to a node in $\mathcal{T}_0$ and is then in S_0. Since there is always an outgoing progressive edge for Steven's vertices and all edges of Audrey's vertices are progressive, we can conclude that S_0 is a trap for Audrey, is a sub-game, and $\mathcal{T}_0$ is a progress measure tree on it. By induction, one gets a Steven attractor decomposition $\mathcal{A}_0$ of S_0, whose tree embeds in $\mathcal{T}_0$.

We have proved that:

$$\mathcal{A} = \langle \varnothing, (S_0, \mathcal{A}_0, A_1), (S_1, \mathcal{A}_1, A_1), \ldots, (S_j, \mathcal{A}_j, A_j) \rangle$$

is a Steven attractor decomposition of $\mathcal{G}$ whose tree embeds in $\mathcal{T}$. □

Lemma 4.3.4 gives one direction of the theorem. For the reverse direction, consider $\mathcal{G}$ a parity game and $\mathcal{A}$ a Steven attractor decomposition of Strahler number k. This decomposition induces a winning strategy for Steven (with exactly one edge going out of any vertex owned by Steven in $\mathcal{G}$). Consider the restriction of $\mathcal{G}$ to this Steven strategy. This is a game where all the vertices belong to Audrey, and which has $\mathcal{A}$ as a Steven attractor decomposition. We can apply Lemma 4.3.4 and obtain a Steven progress measure of Strahler number at most $k+1$. The progress measure thus obtained is also a progress measure of $\mathcal{G}$, which concludes the proof. □

Chapter 5

Strahler universal trees

Having established the equivalence of the Strahler number of a parity game to its Lehtinen number, we shift our attention to tackle Question 4.1.2, which asks if the space complexity of solving register games by Lehtinen [LB20] and Parys [Par20] can be improved.

We give a construction of small Strahler-universal trees that, when used with the progress measure lifting algorithm of Jurdziński and Lazić [Jur00, JL17] or with the Jurdziński-Morvan algorithm [JMT22], yields algorithms that work in quasi-linear space (linear if we exclude poly-logarithmic factors) and quasi-polynomial time. Moreover, usage of our small Strahler-universal trees allows to solve parity games in polynomial time for a wider range of asymptotic settings of the two natural structural complexity parameters (number of priorities d and the Strahler/register number k) than previously known, and that covers as special cases the $k = \mathcal{O}(1)$ criterion of Lehtinen [Leh18] and the $d < \lg n$ and $d = \mathcal{O}(\log n)$ criteriaof Calude et al. [CJK$^+$17], and of Jurdziński and Lazić [JL17], respectively.

Our approach is to develop constructions of small ordered trees into which trees of attractor decompositions or of progress measures can be embedded. Such trees can be seen as natural search spaces for dominion strategies, and existing meta-algorithms such as the Jurdziński-Morvan [JMT22] algorithm and progress measure lifting algorithm [Jur00, JL17] can use them to guide their search, performed in time proportional to the size of the trees in the worst case.

Recall that an ordered tree is *universal* for a class of trees if all trees from the class can be embedded into it. The innovation offered in this chapter is to develop optimised constructions of trees that are universal for classes of trees whose complex structural parameter, such as the Strahler number, is bounded. This is in contrast to less restrictive universal trees introduced by Czerwiński et al. [CDF$^+$19]

and implicitly constructed by Jurdziński and Lazić [JL17], whose sizes therefore grow faster with size parameters, leading to slower algorithms.

Firstly, we give an inductive construction of Strahler-universal trees and an upper bound on their numbers of leaves. Secondly, we provide a succinct bit-string labelling of the Strahler-universal trees, and give an alternative and more explicit characterization of the succinctly-labelled Strahler-universal trees. Thirdly, we argue how the succinct bit-string labelling of Strahler-universal trees facilitates efficient computation of the so-called "level-p successors" in them, which is the key computational primitive that allows using ordered trees to solve parity games. The constructions and techniques we develop here are inspired by and significantly refine those introduced by Jurdziński and Lazić [JL17]. Finally, we also give a lower bound on the size of a Strahler-universal tree, showing that our constructions are optimal and also that efforts in improving Lehtinen's algorithms using Strahler-universal trees fail.

5.1 Strahler-Universal Trees and Their Sizes

Recall that we had define the embedding of an ordered tree in Chapter 2. Also recall that an ordered tree is (n, h)-*universal* [CDF$^+$19] if every (n, h)-small ordered tree can be embedded in it. We define an ordered tree to be k-*Strahler* (n, h)-*universal* if every (n, h)-small ordered tree whose Strahler number is at most k can be embedded in it, and we give a construction of small Strahler-universal trees.

We first give a gentle introduction behind the construction of k-Strahler (n, h)-universal trees ($U^k_{\lg n,h}$). These are constructed with the help of what we call weak k-Strahler (n, h)-universal trees. A tree is said to have Weak Strahler number k if the maximum of the Strahler number of its (strict) subtrees is $k - 1$. Weakly Strahler Universal trees ($V^k_{\lg n,h}$) are trees that can embed any tree with at most n leaves, height at most h, and weak Strahler number at most k.

If a tree has Strahler number $k = 1$ or if it has only one node ($\lg n = 0$), then the k-Strahler (n, h)-universal tree as well as the weak k-Strahler (n, h)-universal tree is just the trivial tree. If instead, the size of the height of the tree is equal to the Strahler number, then the k-Strahler (n, h)-universal tree is just the same as the weakly k-Strahler (n, h)-universal tree. If not, then the weak k-Strahler (n, h)-universal tree would consist of two copies of the weakly k-Strahler $(n/2, h)$-universal tree on either side of a $k-1$-Strahler $(n, h-1)$-universal tree attached to the root (as depicted in Fig. 5.1). This is because there is at most one child of a tree with n leaves will have over $n/2$ leaves and therefore this tree would be able to embed any tree

of weak Strahler number k with n leaves and height h. Moreover, all children have Strahler number at most $k-1$. We finally define the k-Strahler (n,h)-universal tree

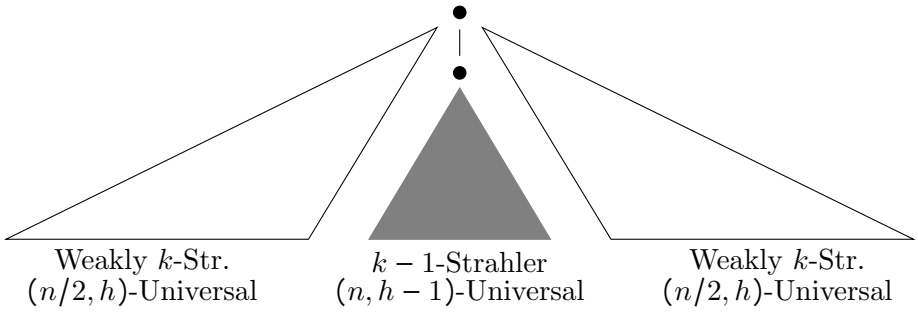


Figure 5.1: Construction of a weakly k-Strahler (n,h)-Universal tree where $h \geqslant k \geqslant 2$ and there are at n is at least 2.

in such cases of $h \geqslant k \geqslant 2$ as the tree obtained by adjoining two copies of the weakly k-Strahler (n,h)-universal trees on either side of a k Strahler $(n,h-1)$-universal tree attached to the root.

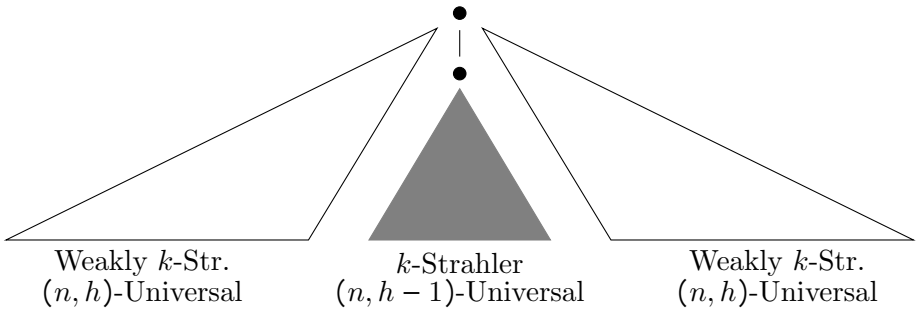


Figure 5.2: Construction of a k-Strahler (n,h)-Universal tree where $h \geqslant k \geqslant 2$ and there are at n is at least 2.

Definition 5.1.1 (Trees $U^k_{t,h}$ and $V^k_{t,h}$)**.** *For all $t \geqslant 0$, we define trees $U^k_{t,h}$ (for all h and k such that $h \geqslant k \geqslant 1$) and $V^k_{t,h}$ (for all h and k such that $h \geqslant k \geqslant 2$) by mutual induction:*

1. *if $h = k = 1$ then $U^k_{t,h} = \langle\rangle$;*
2. *if $h > 1$ and $k = 1$ then $U^k_{t,h} = \left\langle U^k_{t,h-1} \right\rangle$;*
3. *if $h \geqslant k \geqslant 2$ and $t = 0$ then $U^k_{t,h} = V^k_{t,h} = \left\langle U^{k-1}_{t,h-1} \right\rangle$;*
4. *if $h \geqslant k \geqslant 2$ and $t \geqslant 1$ then $V^k_{t,h} = V^k_{t-1,h} \cdot \left\langle U^{k-1}_{t,h-1} \right\rangle \cdot V^k_{t-1,h}$;*

5. *if* $h = k \geqslant 2$ *and* $t \geqslant 1$ *then* $U^k_{t,h} = V^k_{t,h}$;

6. *if* $h > k \geqslant 2$ *and* $t \geqslant 1$ *then* $U^k_{t,h} = V^k_{t,h} \cdot \left\langle U^k_{t,h-1} \right\rangle \cdot V^k_{t,h}$.

Recall that for $g \geqslant 1$, we defined I_g to be the trivial tree, that is the tree with exactly one leaf, of height g. It is routine to verify that if $h \geqslant k = 1$ or $t = 0$ then $U^k_{t,h} = I_h$, and if $h \geqslant k \geqslant 2$ and $t = 0$ then $V^k_{t,h} = I_h$.

Lemma 5.1.2. *For all* $n \geqslant 1$ *and* $h \geqslant k \geqslant 1$*, the ordered tree* $U^k_{\lfloor \lg n \rfloor, h}$ *is* k*-Strahler* (n, h)*-universal.*

Proof. We say that a tree has *weak Strahler number* at most k if every subtree rooted in a child of the root has Strahler number at most $k - 1$. A tree is then *weakly* k*-Strahler* (n, h)*-universal* if every (n, h)-small ordered tree whose weak Strahler number is at most k can be embedded in it. We proceed by induction on the number of leaves in an ordered tree and its height, using the following strengthened inductive hypothesis:

- for all $n \geqslant 1$ and $h \geqslant k \geqslant 1$, ordered tree $U^k_{\lfloor \lg n \rfloor, h}$ is k-Strahler (n, h)-universal;
- for all $n \geqslant 1$ and $h \geqslant k \geqslant 2$, ordered tree $V^k_{\lfloor \lg n \rfloor, h}$ is weakly k-Strahler (n, h)-universal.

Let T be an (n, h)-small ordered tree of Strahler number at most k. If $n = 1$, $h = 1$, or $k = 1$, then T is the trivial tree (with just one leaf) of height at most h, and hence it can be embedded in $U^k_{\lfloor \lg n \rfloor, h} = I_h$, the trivial tree of height h. Likewise, if $h \geqslant k \geqslant 2$ and $n = 1$, then T is the trivial tree of height at most h, and hence it can be embedded in $V^k_{\lfloor \lg n \rfloor, h} = I_h$, the trivial tree of height h.

Otherwise, we have that $T = \langle T_1, \ldots, T_j \rangle$ for some $j \geqslant 1$. We consider two cases: either $\mathrm{Str}(T_i) \leqslant k - 1$ for all $i = 1, \ldots, j$, or there is q such that $\mathrm{Str}(T_q) = k$. Note that the latter case can only occur if $h > k$.

If $\mathrm{Str}(T_i) \leqslant k - 1$ for all $i = 1, \ldots, j$, then we argue that T can be embedded in $V^k_{\lfloor \lg n \rfloor, h}$, and hence also in $U^k_{\lfloor \lg n \rfloor, h}$, because $V^k_{\lfloor \lg n \rfloor, h}$ can be embedded in $U^k_{\lfloor \lg n \rfloor, h}$ by definition (see items 3., 5., and 6. of Definition 5.1.1). Let p (a pivot) be an integer such that both trees $T' = \langle T_1, \ldots, T_{p-1} \rangle$ and $T'' = \langle T_{p+1}, \ldots, T_j \rangle$ are $(\lfloor n/2 \rfloor, h)$-small. Then by the strengthened inductive hypothesis, each of the two trees T' and T'' can be embedded in tree $V^k_{\lfloor \lg \lfloor n/2 \rfloor \rfloor, h} = V^k_{\lfloor \lg n \rfloor - 1, h}$ and tree T_p can be embedded in $U^{k-1}_{\lfloor \lg n \rfloor, h-1}$. It then follows that tree $T = T' \cdot \langle T_p \rangle \cdot T''$ can be embedded in $V^k_{\lfloor \lg n \rfloor, h} = V^k_{\lfloor \lg n \rfloor - 1, h} \cdot \left\langle U^{k-1}_{\lfloor \lg n \rfloor, h-1} \right\rangle \cdot V^k_{\lfloor \lg n \rfloor - 1, h}$.

If $\mathrm{Str}(T_q) = k$ for some q (the pivot), then we argue that T can be embedded in $U^k_{\lfloor \lg n \rfloor, h}$. Note that each of the two trees $T' = \langle T_1, \ldots, T_{q-1} \rangle$ and $T'' =$

$\langle T_{q+1}, \ldots, T_j \rangle$ is (n, h)-small and all trees $T_1, \ldots, T_{q-1}$ and $T_{q+1}, \ldots, T_j$ have Strahler numbers at most $k-1$. By the previous paragraph, it follows that each of the two trees T' and T'' can be embedded in $V^k_{\lfloor \lg n \rfloor, h}$. Moreover, tree T_q is $(n, h-1)$-small and hence, by the inductive hypothesis, it can be embedded in $U^k_{\lfloor \lg n \rfloor, h-1}$. It follows that tree $T = T' \cdot \langle T_q \rangle \cdot T''$ can be embedded in $U^k_{\lfloor \lg n \rfloor, h} = V^k_{\lfloor \lg n \rfloor, h} \cdot \langle U^k_{\lfloor \lg n \rfloor, h-1} \rangle \cdot V^k_{\lfloor \lg n \rfloor, h}$. □

Lemma 5.1.3. *For all $t \geqslant 0$, we have:*

- *if $h \geqslant k = 1$ then* $\operatorname{leaves}\left(U^k_{t,h}\right) = 1$;
- *if $h \geqslant k \geqslant 2$ then* $\operatorname{leaves}\left(U^k_{t,h}\right) \leqslant 2^{t+k}\binom{t+k-2}{k-2}\binom{h-1}{k-1}$.

Proof. The proof is by structural induction, where the inductive hypothesis contains both the statement that for all $t \geqslant 0$ and $h \geqslant k \geqslant 2$, we have:

$$\operatorname{leaves}\left(U^k_{t,h}\right) \leqslant 2^{t+k}\binom{t+k-2}{k-2}\binom{h-1}{k-1}, \tag{5.1}$$

and that for all $t \geqslant 0$ and $h \geqslant k \geqslant 2$, we have the following analogous bound on the number of leaves of trees $V^k_{t,h}$:

$$\operatorname{leaves}\left(V^k_{t,h}\right) \leqslant 2^{t+k-1}\binom{t+k-2}{k-2}\binom{h-2}{k-2}. \tag{5.2}$$

The following cases correspond to the six items in Definition 5.1.1.

1. If $h = k = 1$ then $\operatorname{leaves}\left(U^k_{t,h}\right) = \operatorname{leaves}(\langle\rangle) = 1$.

2. If $h > 1$ and $k = 1$ then a straightforward induction on h can be used to show that $\operatorname{leaves}\left(U^k_{t,h}\right) = 1$.

3. If $h \geqslant k \geqslant 2$ and $t = 0$ then, again, a straightforward induction on h yields that $\operatorname{leaves}\left(V^k_{t,h}\right) = 1 < 2^{t+k-1}\binom{t+k-2}{k-2}\binom{h-2}{k-2}$ and $\operatorname{leaves}\left(U^k_{t,h}\right) = 1 < 2^{t+k}\binom{t+k-2}{k-2}\binom{h-1}{k-1}$.

4. Suppose that $h \geqslant k \geqslant 2$ and $t \geqslant 1$.

   Firstly, for $h \geqslant k = 2$ and $t \geqslant 0$, we slightly strengthen the inductive hypothesis (5.2) to:

   $$\operatorname{leaves}\left(V^2_{t,h}\right) \leqslant 2^{t+1} - 1\,, \tag{5.3}$$

   which we prove by induction on t. Indeed, for $t = 0$ it follows from item 3.

above, and for $t \geqslant 1$, we have:

$$\begin{aligned}\operatorname{leaves}\left(V^2_{t,h}\right) &= \operatorname{leaves}\left(U^1_{t,h-1}\right) + 2 \cdot \operatorname{leaves}\left(V^2_{t-1,h}\right) \\ &\leqslant 1 + 2\left(2^{(t-1)+1} - 1\right) = 2^{t+1} - 1 < 2^{t+1}\binom{t}{0}\binom{h-2}{0},\end{aligned}$$

where the first inequality follows from items 1. or 2. above, and from the strengthened inductive hypothesis (5.3).

Secondly, for $h \geqslant k \geqslant 3$ and $t \geqslant 1$ we have:

$$\begin{aligned}\operatorname{leaves}\left(V^k_{t,h}\right) &= \operatorname{leaves}\left(U^{k-1}_{t,h-1}\right) + 2 \cdot \operatorname{leaves}\left(V^k_{t-1,h}\right) \\ &\leqslant 2^{t+k-1}\binom{t+k-3}{k-3}\binom{h-2}{k-2} + 2 \cdot 2^{t+k-2}\binom{t+k-3}{k-2}\binom{h-2}{k-2} \\ &= 2^{t+k-1}\left[\binom{t+k-3}{k-3} + \binom{t+k-3}{k-2}\right]\binom{h-2}{k-2} \\ &= 2^{t+k-1}\binom{t+k-2}{k-2}\binom{h-2}{k-2},\end{aligned}$$

where the first inequality follows from the inductive hypothesis and the last equality follows from Pascal's identity.

5. Suppose that $h = k \geqslant 2$ and $t \geqslant 1$. Then we have:

$$\begin{aligned}\operatorname{leaves}\left(U^k_{t,h}\right) &= \operatorname{leaves}\left(V^k_{t,h}\right) \leqslant 2^{t+k-1}\binom{t+k-2}{k-2}\binom{h-2}{k-2} \\ &< 2^{t+k}\binom{t+k-2}{k-2}\binom{h-1}{k-1},\end{aligned}$$

where the first inequality follows by the inductive hypothesis and the other one from $h = k$.

6. Suppose $h > k \geqslant 2$ and $t \geqslant 1$. Then we have:

$$\begin{aligned}\operatorname{leaves}\left(U^k_{t,h}\right) &= \operatorname{leaves}\left(U^k_{t,h-1}\right) + 2 \cdot \operatorname{leaves}\left(V^k_{t,h}\right) \\ &\leqslant 2^{t+k}\binom{t+k-2}{k-2}\binom{h-2}{k-1} + 2 \cdot 2^{t+k-1}\binom{t+k-2}{k-2}\binom{h-2}{k-2} \\ &= 2^{t+k}\binom{t+k-2}{k-2}\left[\binom{h-2}{k-1} + \binom{h-2}{k-2}\right] = 2^{t+k}\binom{t+k-2}{k-2}\binom{h-1}{k-1},\end{aligned}$$

where the first inequality follows from the inductive hypothesis and the last equality follows from Pascal's identity. □.

Theorem 5.1.4. *For $k \leqslant \lg n$, the number of leaves of the k-Strahler (n,h)-universal ordered trees $U^k_{\lfloor \lg n \rfloor, h}$ is $n^{\mathcal{O}(1)} \cdot (h/k)^k = n^{k \lg(h/k)/\lg n + \mathcal{O}(1)}$, which is polynomial in n if $k \cdot \lg(h/k) = \mathcal{O}(\log n)$. In more detail, the number is at most $n^{c(n)} \cdot (h/k)^k$, where $c(n) = 5.45$ if $k \leqslant \lg n$, $c(n) = 3 + o(1)$ if $k = \mathcal{O}(\log n)$, and $c(n) = 1 + o(1)$ if $k = \mathcal{O}(1)$.*

Remark 2. *By Proposition 4.0.1 and Lemma 5.1.2, for all positive integers n and h, the tree $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is (n,h)-universal. Theorem 5.1.4 implies that the number of leaves of $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is $n^{\lg(h/\lg n) + \mathcal{O}(1)}$, which matches the asymptotic number of leaves of (n,h)-universal trees of Jurdziński and Lazić [JL17, Lemma 6]. In particular, if $h = \mathcal{O}(\log n)$ then $\lg(h/\lg n) = \mathcal{O}(1)$, and hence the number of leaves of $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is polynomial in n.*

Proof of Theorem 5.1.4. By Lemma 5.1.2, ordered tree $U^k_{\lfloor \lg n \rfloor, h}$ is k-Strahler (n,h)-universal. By Lemma 5.1.3, its number of leaves is at most $2^{\lfloor \lg n \rfloor + k} \binom{\lfloor \lg n \rfloor + k - 2}{k-2} \binom{h-1}{k-1}$.

We analyze in turn the three terms $2^{\lfloor \lg n \rfloor + k}$, $\binom{\lfloor \lg n \rfloor + k - 2}{k-2}$, and $\binom{h-1}{k-1}$. Firstly, we note that

$$2^{\lfloor \lg n \rfloor + k} = \mathcal{O}\left(n^{p_1(n,k)}\right)$$

where $p_1(n,k) = 1 + k/\lg n$, because $2^k = n^{k/\lg n}$. Secondly, $k \leqslant \lg n$ implies that $\lfloor \lg n \rfloor + k - 2 < 2 \lg n$, therefore we have

$$\binom{\lfloor \lg n \rfloor + k - 2}{k-2} < 2^{2 \lg n} = n^2$$

and hence

$$\binom{\lfloor \lg n \rfloor + k - 2}{k-2} = \mathcal{O}(n^{p_2(n,k)})$$

where $p_2(n,k) \leqslant 2$. Thirdly, applying the inequality $\binom{i}{j} \leqslant (ei/j)^j$ to the binomial coefficient $\binom{h}{k}$, we obtain $\binom{h-1}{k-1} < \binom{h}{k} \leqslant (eh/k)^k = 2^{k \lg(eh/k)}$, and hence

$$\binom{h-1}{k-1} = \mathcal{O}(n^{p_3(n,h,k)})$$

where

$$p_3(n,h,k) = k \lg(eh/k)/\lg n = k \lg(h/k)/\lg n + k \lg e/\lg n.$$

Note that if we let $p(n,h,k) = p_1(n,k) + p_2(n,k) + p_3(n,h,k)$ then the number of leaves in trees $U^k_{\lfloor \lg n \rfloor, h}$ is $\mathcal{O}\big(n^{p(n,h,k)}\big)$. Since $k \leqslant \lg n$ implies $k/\lg n \leqslant 1$ and $k \lg e/\lg n \leqslant \lg e$, we obtain

$$p(n,h,k) \leqslant k \lg(h/k)/\lg n + 4 + \lg e < k \lg(h/k)/\lg n + 5.45$$

and hence the number of leaves in trees $U^k_{\lfloor \lg n \rfloor, h}$ is

$$n^{k \lg(h/k)/\lg n + \mathcal{O}(1)}.$$

If we further assume that $k = \mathcal{O}(\log n)$ then the constant 5.45 can be straightfowardly reduced to $3 + \mathcal{O}(1)$ because then $k/\lg n$ and $k \lg e/\lg n$ are $\mathcal{O}(1)$. Moreover, the estimate

$$\binom{\lfloor \lg n \rfloor + k - 2}{k-2} = \mathcal{O}(n^2)$$

can be improved with further assumptions about k as a function of n; for example, if $k = \mathcal{O}(1)$ then $\binom{\lfloor \lg n \rfloor + k - 2}{k-2}$ is only polylogarithmic in n and hence $\binom{\lfloor \lg n \rfloor + k - 2}{k-2}$ is $n^{\mathcal{O}(1)}$, bringing $3 + o(1)$ down to $1 + o(1)$. □

5.2 Labelled Strahler-Universal Trees.

Recall the bit-string ordering on $\mathbb{W} = \{0,1\}^*$. For a bit string $\beta \in \mathbb{W}$, we write $|\beta|$ for the number of bits in the string. For example, we have $|\varepsilon| = 0$ and $|010| = 3$, and $|11| = 2$. Suppose that $\langle \beta_i, \beta_{i-1}, \ldots, \beta_1 \rangle$ is a node in a $\mathbb{W}$-labelled ordered tree. Then if $\beta_j = b\beta$ for some $j = 1, 2, \ldots, i$, $b \in \{0,1\}$, and $\beta \in \mathbb{W}$, then we refer to the first bit b as the *leading bit* in β_j, and we refer to all the following bits in β as *non-leading bits* in β_j. For example, node $\langle \varepsilon, 010, \varepsilon, \varepsilon, 11 \rangle$ has two non-empty strings and hence two leading bits, and it uses three non-leading bits overall, because $|010| + |11| - 2 = 3$.

For a bit $b \in \{0,1\}$ and $\mathcal{L} = \langle (\beta_1, \mathcal{L}_1), \ldots, (\beta_\ell, \mathcal{L}_\ell) \rangle$, a $\mathbb{W}$-labelled ordered tree , we define $[\mathcal{L}]^b$ to be the $\mathbb{W}$-labelled ordered tree $\mathcal{L} = \langle (b\beta_1, \mathcal{L}_1), \ldots, (b\beta_\ell, \mathcal{L}_\ell) \rangle$. In other words, $[\mathcal{L}]^b$ is the labelled ordered tree that is obtained from $\mathcal{L}$ by adding an extra copy of bit b as the leading bit in the labels of all children of the root of $\mathcal{L}$.

The inductive structure of the next definition is identical to that of Definition 5.1.1, and hence labelled ordered trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$ defined here are labellings of the ordered trees $U^k_{t,h}$ and $V^k_{t,h}$, respectively.

Definition 5.2.1 (Trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$)**.** *For all $t \geqslant 0$, we define $\mathbb{W}$-labelled ordered*

trees $\mathcal{U}^k_{t,h}$ *(for all* h *and* k *such that* $h \geqslant k \geqslant 1$*) and* $\mathcal{V}^k_{t,h}$ *(for all* h *and* k *such that* $h \geqslant k \geqslant 2$*) by mutual induction:*

1. *if* $h = k = 1$ *then* $\mathcal{U}^k_{t,h} = \langle\rangle$*;*
2. *if* $h > 1$ *and* $k = 1$ *then* $\mathcal{U}^k_{t,h} = \left\langle\left(\varepsilon, \mathcal{U}^k_{t,h-1}\right)\right\rangle$*;*
3. *if* $h \geqslant k \geqslant 2$ *and* $t = 0$ *then* $\mathcal{V}^k_{t,h} = \left\langle\left(\varepsilon, \mathcal{U}^{k-1}_{t,h-1}\right)\right\rangle$ *and* $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h}\right]^0 = \left\langle\left(0, \mathcal{U}^{k-1}_{t,h-1}\right)\right\rangle$*;*
4. *if* $h \geqslant k \geqslant 2$ *and* $t \geqslant 1$ *then* $\mathcal{V}^k_{t,h} = \left[\mathcal{V}^k_{t-1,h}\right]^0 \cdot \left\langle\left(\varepsilon, \mathcal{U}^{k-1}_{t,h-1}\right)\right\rangle \cdot \left[\mathcal{V}^k_{t-1,h}\right]^1$*;*
5. *if* $h = k \geqslant 2$ *and* $t \geqslant 1$ *then* $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h}\right]^0$*;*
6. *if* $h > k \geqslant 2$ *and* $t \geqslant 1$ *then* $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h}\right]^0 \cdot \left\langle\left(\varepsilon, \mathcal{U}^k_{t,h-1}\right)\right\rangle \cdot \left[\mathcal{V}^k_{t,h}\right]^1$*.*

The inductive definition of labelled ordered trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$ makes it straightforward to argue that their unlabellings are equal to trees $U^k_{t,h}$ and $V^k_{t,h}$, respectively, and hence to transfer to them the Strahler-universality established in Lemma 5.1.2 and upper bounds on the numbers of leaves established in Lemma 5.1.3 and Theorem 5.1.4. We now give an alternative and more explicit characterization of those trees, which will be more suitable for algorithmic purposes. To that end, we define $\mathbb{W}$-labelled trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$ and then we argue that they are equal to trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$, respectively, by showing that they satisfy all the recurrences in Definition 5.2.1.

Definition 5.2.2 (Trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$)**.** *For all* $t \geqslant 0$ *and* $h \geqslant k \geqslant 1$*, we define* $\mathbb{W}$*-labelled ordered trees* $\mathcal{B}^k_{t,h}$ *as the tree generated by sequences* $\langle\beta_{h-1}, \ldots, \beta_1\rangle$ *such that:*

1. *the number of non-empty bit strings among* β_{h-1}*,* $\ldots$*,* β_1 *is* $k-1$*;*
2. *the number of bits used in bit strings* β_{h-1}*,* $\ldots$*,* β_1 *overall is at most* $(k-1)+t$*;*

and for every $i = 1, \ldots, h-1$*, we have the following:*

3. *if there are less than* $k-1$ *non-empty bit strings among* β_{h-1}*,* $\ldots$*,* β_{i+1}*, but there are* t *non-leading bits used in them, then* $\beta_i = 0$*;*
4. *if all bit strings* β_i*,* $\ldots$*,* β_1 *are non-empty, then each of them has* 0 *as its leading bit.*

For all $t \geqslant 0$ *and* $h \geqslant k \geqslant 2$*, we define* $\mathbb{W}$*-labelled ordered trees* $\mathcal{C}^k_{t,h}$ *as the tree generated by sequences* $\langle\beta_{h-1}, \ldots, \beta_1\rangle$ *such that:*

1. *the number of non-empty bit strings among* $\beta_{h-2}, \ldots, \beta_1$ *is* $k-2$*;*

2. *the number of bits used in bit strings* $\beta_{h-1}, \ldots, \beta_1$ *overall is at most* $(k-2)+t$*;*

and for every $i = 1, \ldots, h-1$*, we have the following:*

3. *if there are less than* $k-2$ *non-empty bit strings among* $\beta_{h-2}, \ldots, \beta_{i+1}$*, but there are* $t - |\beta_{h-1}|$ *non-leading bits used in them, then* $\beta_i = 0$*;*

4. *if all bit strings* $\beta_i, \ldots, \beta_1$ *are non-empty, then each of them has* 0 *as its leading bit.*

Lemma 5.2.3. *For all* $t \geqslant 0$ *and* $h \geqslant k \geqslant 1$*, we have* $\mathcal{U}^k_{t,h} = \mathcal{B}^k_{t,h}$*.*

The following corollary follows from Lemma 5.2.3, and from the identical inductive structures of Definitions 5.1.1 and 5.2.1.

Corollary 5.2.3.1. *For all* $t \geqslant 0$ *and* $h \geqslant k \geqslant 1$*, the unlabelling of* $\mathcal{B}^k_{t,h}$ *is equal to* $U^k_{t,h}$*.*

The next proposition formalizes the following non-rigorous interpretation of the difference between trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$:

- If a sequence $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a node in $\mathcal{B}^k_{t,h}$ then the bit string β_{h-1} can be either empty or non-empty, and if it is non-empty then its first bit is the leading bit.

- On the other hand, if a sequence $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a node in $\mathcal{C}^k_{t,h}$ then the bit string β_{h-1} is always to be understood as non-empty. It can be thought of as obtained by removal of its "original" leading bit in the corresponding leaf in tree $\mathcal{B}^k_{t,h}$, and hence it consists only of (possibly zero) non-leading bits.

Proposition 5.2.4. *For all* $t \geqslant 1$ *and* $h \geqslant k \geqslant 2$*, we have:*

1. *if* $h = k$ *then* $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{C}^k_{t,h}$ *if and only if* $\langle 0\beta_{h-1}, \beta_{h-2}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{B}^k_{t,h}$*;*

2. *if* $h > k$ *then for both* $b \in \{0, 1\}$*, we have that* $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{C}^k_{t,h}$ *if and only if* $\langle b\beta_{h-1}, \beta_{h-2}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{B}^k_{t,h}$*;*

3. $\langle \varepsilon, \beta_{h-2}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{C}^k_{t,h}$ *if and only if* $\langle \beta_{h-2}, \ldots, \beta_1 \rangle$ *is a leaf in* $\mathcal{B}^{k-1}_{t,h-1}$*.*

Proof of Lemma 5.2.3. We argue that trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$ satisfy all the recurrences in Definition 5.2.1 that involve trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$, respectively.

1. If $h = k = 1$ then tree $\mathcal{B}^k_{t,h}$ is the trivial tree $\langle\rangle$.

2. If $h > k = 1$ then $\mathcal{B}^k_{t,h}$ has only one leaf $\left\langle \varepsilon^{h-1} \right\rangle$, and hence we have $\mathcal{B}^k_{t,h} = \left\langle \left(\varepsilon, \mathcal{B}^k_{t,h-1} \right) \right\rangle$.

3. Suppose that $h \geqslant k \geqslant 2$ and $t = 0$. Then $\mathcal{B}^k_{t,h}$ has exactly one leaf, which is of the form $\left\langle 0^{k-1}, \varepsilon^{h-k} \right\rangle$, and $\mathcal{C}^k_{t,h}$ has exactly one leaf, which is of the form $\left\langle \varepsilon, 0^{k-2}, \varepsilon^{h-k} \right\rangle$. It follows that $\mathcal{C}^k_{t,h} = \left\langle \left(\varepsilon, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle$ and $\mathcal{B}^k_{t,h} = \left[\mathcal{C}^k_{t,h} \right]^0 = \left\langle \left(0, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle$.

4. Suppose that $h \geqslant k \geqslant 2$ and $t \geqslant 1$. We argue that the following recurrence holds:

$$\mathcal{C}^k_{t,h} = \left[\mathcal{C}^k_{t-1,h} \right]^0 \cdot \left\langle \left(\varepsilon, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle \cdot \left[\mathcal{C}^k_{t-1,h} \right]^1 .$$

   First, we show that every leaf in $\mathcal{C}^k_{t,h}$ is also a leaf in tree $\left\langle \left(\varepsilon, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle$ or in tree $\left[\mathcal{C}^k_{t-1,h} \right]^b$ for some $b \in \{0, 1\}$. Suppose that $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t,h}$.

   - If $\beta_{h-1} = \varepsilon$ then $\langle \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{B}^{k-1}_{t,h-1}$, and hence the node $\ell = \langle \varepsilon, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in tree $\left\langle \left(\varepsilon, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle$.
   - If $\beta_{h-1} = b\beta$ for some $b \in \{0, 1\}$ then $\langle \beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t-1,h}$, and hence $\ell = \langle b\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\left[\mathcal{C}^k_{t-1,h} \right]^b$.

   Conversely, we now argue that if $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in labelled ordered tree $\left\langle \left(\varepsilon, \mathcal{B}^{k-1}_{t,h-1} \right) \right\rangle$, then it is also a leaf in $\mathcal{C}^k_{t,h}$. Note that the premise implies that $\beta_{h-1} = \varepsilon$ and $\langle \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{B}^{k-1}_{t,h-1}$, and hence, by item 3. in Proposition 5.2.4, we have that $\ell = \langle \varepsilon, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\mathcal{C}^k_{t,h}$.

   Finally, we argue that if $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in a tree $\left[\mathcal{C}^k_{t-1,h} \right]^b$ for $b \in \{0, 1\}$, then it is also a leaf in $\mathcal{C}^k_{t,h}$. Indeed, the premise implies that $\beta_h = b\beta$ and $\langle \beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t-1,h}$, and hence $\ell = \langle b\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\mathcal{C}^k_{t,h}$.

5. Suppose that $h = k \geqslant 2$ and $t \geqslant 1$. We argue that then we have $\mathcal{B}^k_{t,h} = \left[\mathcal{C}^k_{t,h} \right]^0$.

   First, let $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ be a leaf in tree $\mathcal{B}^k_{t,h}$. Since $h = k$, all bit strings $\beta_{h-1}, \ldots, \beta_1$ are non-empty, and hence $\beta_{h-1} = 0\beta$ for some $\beta \in \mathbb{W}$. By item 1. of Proposition 5.2.4, it follows that the sequence $\langle \beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t,h}$, and hence $\ell = \langle 0\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\left[\mathcal{C}^k_{t,h} \right]^0$.

Conversely, let $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ be a leaf in tree $\left[\mathcal{C}^k_{t,h}\right]^0$. Then $\beta_{h-1} = 0\beta$ for some $\beta \in \mathbb{W}$ and sequence $\langle \beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t,h}$. By item 1. of Proposition 5.2.4, it follows that $\ell = \langle 0\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\mathcal{B}^k_{t,h}$.

6. Suppose that $h > k \geqslant 2$ and $t \geqslant 1$. We argue that then the following recurrence holds:

$$\mathcal{B}^k_{t,h} = \left[\mathcal{C}^k_{t,h}\right]^0 \cdot \left\langle \left(\varepsilon, \mathcal{B}^k_{t,h-1}\right) \right\rangle \cdot \left[\mathcal{C}^k_{t,h}\right]^1 .$$

First, we show that every leaf in $\mathcal{B}^k_{t,h}$ is also a leaf in tree $\left\langle \left(\varepsilon, \mathcal{B}^k_{t,h-1}\right) \right\rangle$ or in tree $\left[\mathcal{C}^k_{t,h}\right]^b$ for some $b \in \{0,1\}$. Suppose that $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{B}^k_{t,h}$.

- If $\beta_{h-1} = \varepsilon$ then $\langle \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{B}^k_{t,h-1}$, and hence the node $\ell = \langle \varepsilon, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\left\langle \left(\varepsilon, \mathcal{B}^k_{t,h-1}\right) \right\rangle$.
- If $\beta_{h-1} = b\beta$ for some $b \in \{0,1\}$ then, by item 2. of Proposition 5.2.4, $\langle \beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t,h}$, and hence $\ell = \langle b\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\left[\mathcal{C}^k_{t,h}\right]^b$.

Conversely, we now argue that if $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in labelled ordered tree $\left\langle \left(\varepsilon, \mathcal{B}^k_{t,h-1}\right) \right\rangle$, then it is also a leaf in $\mathcal{B}^k_{t,h}$. Note that the premise implies that $\beta_{h-1} = \varepsilon$ and $\langle \beta_{h-2}, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{B}^k_{t,h-1}$. It follows that $\ell = \langle \varepsilon, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\mathcal{B}^k_{t,h}$.

Finally, we argue that if $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ is a leaf in $\left[\mathcal{C}^k_{t,h}\right]^b$ for some $b \in \{0,1\}$, then it is also a leaf in $\mathcal{B}^k_{t,h}$. The premise implies that $\beta_{h-1} = b\beta$ for some $\beta \in \mathbb{W}$ and that $\langle \beta, \ldots, \beta_1 \rangle$ is a leaf in $\mathcal{C}^k_{t,h}$. By item 2. of Proposition 5.2.4, it follows that $\ell = \langle b\beta, \beta_{h-2}, \ldots, \beta_1 \rangle$ is indeed a leaf in $\mathcal{B}^k_{t,h}$.

Straightforward structural induction (on the structure of labelled ordered trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$) yields that $\mathcal{B}^k_{t,h} = \mathcal{U}^k_{t,h}$ and $\mathcal{C}^k_{t,h} = \mathcal{V}^k_{t,h}$. □

5.3 Efficiently navigating labelled Strahler-universal trees

The computation of the *level-p successor* of a leaf in a labelled ordered tree of height h is the following problem: given a leaf $\langle \beta_h, \beta_{h-1}, \ldots, \beta_1 \rangle$ in the tree and given a number p, such that $1 \leqslant p \leqslant h$, compute the $<_{\text{lex}}$-smallest leaf $\left\langle \beta'_h, \beta'_{h-1}, \ldots, \beta'_1 \right\rangle$ in the tree, such that $\left\langle \beta_h, \ldots, \beta_p \right\rangle <_{\text{lex}} \left\langle \beta'_h, \ldots, \beta'_p \right\rangle$. As (implicitly) explained by Jurdziński and Lazić [JL17, Proof of Theorem 7], the level-p successor computation is the key primitive used extensively in an implementation of a progress measure lifting algorithm.

Lemma 5.3.1. *Every leaf in tree $\mathcal{B}^k_{t,h}$ can be represented using $\mathcal{O}((k+t)\log h)$ bits and for every $p = 1, 2, \ldots, h$, the level-p successor of a leaf in tree $\mathcal{B}^k_{t,h}$ can be computed in time $\mathcal{O}((k+t)\log h)$.*

Proof. Consider the following representation of a leaf $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ in $\mathcal{B}^k_{t,h}$: for each of the at most $k+t$ bits used in the bit strings $\beta_{h-1}, \ldots, \beta_1$ overall, store the value of the bit itself and the number, written in binary, of the component in the h-tuple that this bit belongs to. Altogether, the number of bits needed is $\mathcal{O}((k+t)\cdot(1+\lg h)) = \mathcal{O}((k+t)\log h)$.

We now consider computing the level-p successor of a leaf $\ell = \langle \beta_{h-1}, \ldots, \beta_1 \rangle$ in tree $\mathcal{B}^k_{t,h}$. We split the task of computing the level-p successor ℓ' of leaf ℓ into the following two steps:

- find the lowest ancestor $\langle \beta_{h-1}, \ldots, \beta_q \rangle$ of $\langle \beta_{h-1}, \ldots, \beta_p \rangle$ (that is, smallest q satisfying $q \geqslant p$) that has the next sibling $\langle \beta_{h-1}, \ldots, \beta_{q+1}, \beta'_q \rangle$ in $\mathcal{B}^k_{t,h}$;
- find the smallest leaf $\ell' = \langle \beta_{h-1}, \ldots, \beta_{q+1}, \beta'_q, \beta'_{q-1}, \ldots, \beta'_1 \rangle$ that is a descendant of node $\langle \beta_{h-1}, \ldots, \beta_{q+1}, \beta'_q \rangle$ in $\mathcal{B}^k_{t,h}$.

For node $\ell_r = \langle \beta_{h-1}, \ldots, \beta_r \rangle$, where $q \leqslant r \leqslant h-1$, we can determine whether it has the next sibling $\ell'_r = \langle \beta_{h-1}, \ldots, \beta_{r+1}, \beta'_r \rangle$ in $\mathcal{B}^k_{t,h}$ and find it, by considering the following cases. Firstly, we identify the cases in which ℓ_r does not have the next sibling:

- the number of non-empty strings among $\beta_{h-1}, \ldots, \beta_{r+1}$ is $k-1$;
- the number of non-leading bits used in strings $\beta_{h-1}, \ldots, \beta_{r+1}$ is t;
- $\beta_r = 01^j$ for some $j \geqslant 0$, the number of non-leading bits used in strings $\beta_{h-1}, \ldots, \beta_r$ is t, and all bit strings $\beta_r, \ldots, \beta_1$ are non-empty;
- $\beta_r = 1^j$ for some $j \geqslant 1$, and the number of non-leading bits used in strings $\beta_{h-1}, \ldots, \beta_r$ is t.

Define k_{r+1} to be equal to $k-1$ minus the number of non-empty bit strings among $\beta_{h-1}, \ldots, \beta_{r+1}$, and define t_{r+1} to be equal to t minus the number of non-leading bits used in strings $\beta_{h-1}, \ldots, \beta_{r+1}$. We note that the subtree of $\mathcal{B}^k_{t,h}$ that is rooted at node ℓ_{r+1} is a copy of tree $\mathcal{B}^{k_{r+1}}_{t_{r+1},r+1}$. Recall that trees $\mathcal{B}^k_{t,h}$ satisfy the same recurrences as trees $\mathcal{U}^k_{t,h}$. Observe that the four cases above capture ℓ_r being the largest child of the root of the copy of $\mathcal{B}^{k_{r+1}}_{t_{r+1},r+1}$ rooted in node ℓ_{r+1} in $\mathcal{B}^k_{t,h}$, that correspond to items 2., 3., 5., and 6. of Definition 5.2.1, respectively.

Secondly, we consider the remaining two cases in which ℓ_r does have the next sibling and we show how to find it by setting the value of β'_r accordingly.

- If less than t non-leading bits are used in strings $\beta_{h-1}, \ldots, \beta_r$ then set $\beta_r' = \beta_r 10^j$ for some $j \geqslant 0$, so that exactly t non-leading bits are used in strings $\beta_{h-1}, \ldots, \beta_{r+1}, \beta_r'$.
- If exactly t non-leading bits are used in strings $\beta_{h-1}, \ldots, \beta_r$, and $\beta_r = \beta 01^j$ for some $\beta \in \mathbb{W}$ and $j \geqslant 0$, then set $\beta_r' = \beta$.

Finally, we set $\ell' = \left\langle \beta_{h-1}, \ldots, \beta_{q+1}, \beta_q', 00^i, 0, \ldots, 0, \varepsilon, \ldots, \varepsilon \right\rangle$ for some suitable $i \geqslant 0$, so as to make the number of non-empty bit strings in ℓ' equal to $k-1$, and the number of bits used in all the bit strings in ℓ' equal to $(k-1)+t$.

To argue that the above case analyses can be implemented to work in time $\mathcal{O}((k+t)\log h)$, while using the succinct representation described above, is tedious and hence we eschew it. □

5.4 Lower bound for Strahler-universal trees

Theorem 5.4.1. *The size of a k-Strahler (n,h)-universal tree is $\Omega\Big(n \lg n \left(\frac{h}{k-1}\right)^{k-1}\Big)$.*

Proof. We show that there exists a lower bound on the size of a k-Strahler (n,h)-universal tree, denoted by $f(n,h,k)$, such that:

$$f(n,h,k) \geqslant \frac{n \lg n}{2048}\binom{h}{k-1} > \frac{n \lg n}{2048}\left(\frac{h}{k-1}\right)^{k-1}$$

More precisely, we show that:

$$f(n,h,k) \geqslant \max\left(\frac{n \lg n}{2048}\binom{h}{k-1}, \binom{h}{k-1}\right) \tag{5.4}$$

This proof closely follows the proof of Czerwiński et al's [CDF+19] lower bound.

First, define $f(n,h,k)$ in the following cases:

- $f(1,h,k) = 1$, for all h, k,
- $f(n,h,1) = 1$, for all n, h, and
- $f(n,1,k) = n$ for all n and $k \geqslant 2$

Clearly, $f(n,h,k)$ is a lower bound on the size of a k-Strahler (n,h)-universal tree and satisfies Equation 5.4.

We will show that one can define by induction $f(n,h,k)$ as a lower bound on the size of a k-Strahler (n,h)-universal tree satisfying Equation 5.4 for $k \leqslant \lg n$ and $k \leqslant h$.

Then, by defining $f(n,h,k)$ as follows in the remaining cases, we can conclude the proof:

- $f(n,h,k) = f(n,h,\lg n)$ if $k > \lg n$,
- $f(n,h,k) = f(n,h,h)$ if $k > h$,

We now define $f(n,h,k)$ by induction, under the assumption that $k \leqslant \lg n$ and $k \leqslant h$. For a k-Strahler (n,h)-universal tree T, we claim that the number of nodes at depth $h-1$ with degree $\geqslant \delta$ is at least

- $f(n,h-1,k)$ if $\delta = 1$ and
- $f(\lfloor n/\delta \rfloor, h-1, k-1)$ for $\delta > 1$.

Suppose $\delta = 1$, the bound is obvious. For $\delta > 1$, consider any k-Strahler (n,h)-universal tree U. Let U_δ be the tree obtained by deleting all nodes that do not have degree at least δ. Consider any tree T_δ with at most $\lfloor n/\delta \rfloor$ many nodes, height at most h and Strahler number at most $k-1$. We will show that this tree T_δ must embed into U_δ, showing that U_δ is $(k-1)$-Strahler $(\lfloor n/\delta \rfloor, h-1)$-universal. In T_δ, add δ many children to each of the $\lfloor n/\delta \rfloor$ leaves. This results in a tree that has Strahler number at most k and at most n children and must then embed into U_δ.

Since the number of vertices at depth $h-1$ with degree δ is as argued, we can conclude that there exists a lower bound $f(n,h,k)$ on the size of a k-Strahler (n,h)-universal tree such that:

$$f(n,h,k) \geqslant f(n,h-1,k) + \sum_{\delta=2}^{n} f(\lfloor n/\delta \rfloor, h-1, k-1). \tag{5.5}$$

By induction, we get:

$$f(n,h,k) \geqslant \frac{n \lg n}{2048}\binom{h-1}{k-1} + \frac{1}{2048}\sum_{\delta=2}^{n} \lfloor n/\delta \rfloor \lg \lfloor n/\delta \rfloor \binom{h-1}{k-2}.$$

This gives:

$$f(n,h,k) \geqslant \frac{n \lg n}{2048}\binom{h-1}{k-1} + \frac{1}{2048}\binom{h-1}{k-2}\sum_{\delta=2}^{n} \lfloor n/\delta \rfloor \lg \lfloor n/\delta \rfloor.$$

Case 1. We will show that for $n > 2^8$ and $h, k > 1$,

$$\sum_{\delta=2}^{n} \lfloor n/\delta \rfloor \lg \lfloor n/\delta \rfloor \geqslant n \lg n. \tag{5.6}$$

Let $p = \lfloor \lg n \rfloor$, then

$$\sum_{\delta=2}^{n} \lfloor n/\delta \rfloor \lg \lfloor n/\delta \rfloor \geqslant \sum_{i=1}^{p} 2^{i-1} \left(\left\lfloor \frac{2^p}{2^i} \right\rfloor \lg \left\lfloor \frac{2^p}{2^i} \right\rfloor \right)$$
$$= \sum_{i=1}^{p} (p-i) 2^{p-1} = 2^{p+1} \sum_{i=1}^{p} \frac{p-i}{4}.$$

Since $2^{p+1} \geqslant n$ and $\sum_{i=1}^{p} \frac{p-i}{4} = \frac{p(p-1)}{8} \geqslant \lg n$ for all $p > 8$, (5.6) holds for all $n > 2^8$.

By (5.6) and Pascal's identity, we finally get for all $n > 2^8$:

$$f(n,h,k) \geqslant \frac{n \lg n}{2048} \binom{h}{k-1} \geqslant \max\left(\binom{h}{k-1}, \frac{n \lg n}{2048} \binom{h}{k-1} \right).$$

Case 2. It remains to handle the case $n \leqslant 2^8$. For $2 \leqslant n \leqslant 2^8$ observe that

$$\binom{h}{k-1} \geqslant \frac{n \lg n}{2048} \binom{h}{k-1}.$$

It is then enough to prove by induction that

$$f(n,h,k) \geqslant \binom{h}{k-1}.$$

We get the following sequence of inequalities - the first is from equation (5.5), the second is by induction hypothesis and the last one from Pascal's identity:

$$f(n,h,k) \geqslant f(n,h-1,k) + \sum_{\delta=2}^{n} f(\lfloor n/\delta \rfloor, h-1, k-1)$$
$$\geqslant \binom{h-1}{k-1} + \binom{h-1}{k-2} = \binom{h}{k-1}. \square$$

5.5 Strahler-universal progress measure lifting algorithm

Jurdziński and Lazić [JL17, Section IV] have implicitly suggested that the progress-measure lifting algorithm [Jur00] can be run on any ordered tree and they have established the correctness of such an algorithm if their Jurdziński-Lazić universal trees were used. This has been further clarified by Czerwiński et al. [CDF+19, Section 2.3], who have explicitly argued that any $(n, d/2)$-universal ordered tree is sufficient to solve an (n,d)-small parity game in this way. We make explicit a

more detailed observation that follows using the same standard arguments (see, for example, Jurdziński and Lazić [JL17, Theorem 5]).

Proposition 5.5.1. *Suppose the progress measure-lifting algorithm is run on a parity game $\mathcal{G}$ and on an ordered tree T. Let D be the largest Steven dominion in $\mathcal{G}$ on which there is a Steven progress measure whose tree can be embedded in T. Then the algorithm returns a Steven dominion strategy on D.*

An elementary corollary of this observation is that if the progress-measure lifting algorithm is run on the tree of a progress measure on some Steven dominion in a parity game, then the algorithm produces a Steven dominion strategy on a superset of that dominion. Note that this is achieved in polynomial time because the tree of a progress measure on an (n, d)-small parity game is $(n, d/2)$-small and the running time of the algorithm is dominated by the size of the tree [JL17, Section IV.B].

Theorem C. *There is an algorithm for solving parity games with n vertices, d priorities, and of Strahler number k in quasi-linear space and time $n^{\mathcal{O}(1)} \cdot (d/2k)^k = n^{k \lg(d/k)/\lg n + \mathcal{O}(1)}$, which is polynomial in n if $k \cdot \lg(d/k) = \mathcal{O}(\log n)$.*

Proof. By Proposition 4.0.1, we may assume that $k \leqslant 1 + \lg n$. In order to solve an (n, d)-small parity game of Steven Strahler number k, run the progress-measure lifting algorithm for Steven on tree $\mathcal{B}^{k+1}_{\lfloor \lg n \rfloor, d/2+1}$, which is $(k+1)$-Strahler $(n, d/2+1)$-universal by Lemma 5.1.2 and Corollary 5.2.3.1. By Theorem 4.3.2 and by Proposition 5.5.1, the algorithm will then return a Steven dominion strategy on the largest Steven dominion. The running time and space upper bounds follow from Theorem 5.1.4, by the standard analysis of progress-measure lifting as in [JL17, Theorem 7], and by Lemma 5.3.1. □

5.6 Remarks

We highlight the $k \cdot \lg(d/k) = \mathcal{O}(\log n)$ criterion from Theorem C as offering a novel trade-off between two natural structural complexity parameters of parity games (number of priorities d and the Strahler/Lehtinen number k) that enables solving them in time that is polynomial in the number of vertices n. It includes as special cases both the $d < \lg n$ criterion mentioned by Calude et al. [CJK$^+$22, Theorem 16] and the $d = \mathcal{O}(\log n)$ criterion of Jurdziński and Lazić [JL17, Theorem 7] (set $k = \lfloor \lg n \rfloor + 1$ and use Propositions 2.0.4 and 4.0.1 to justify it), and the $k = \mathcal{O}(1)$ criterion of Lehtinen and Boker [LB20, Theorem 4.2] (by Theorem B).

We argue that the new $k \cdot \lg(d/k) = \mathcal{O}(\log n)$ criterion (Theorem C) enabled by our results (coincidence of the Strahler and the Lehtinen numbers: Theorem B) and techniques (small and efficiently navigable Strahler-universal trees: Theorem 5.1.4, Corollary 5.2.3.1, and Lemma 5.3.1) considerably expands the asymptotic ranges of the natural structural complexity parameters in which parity games can be solved in polynomial time. We illustrate it by considering the scenario in which the rates of growth of both k and $\lg d$ as functions of n are $\mathcal{O}\left(\sqrt{\log n}\right)$, i.e., d is $2^{\mathcal{O}\left(\sqrt{\log n}\right)}$. Note that the number of priorities d in this scenario is allowed to grow as fast as $2^{b \cdot \sqrt{\lg n}}$ for an arbitrary positive constant b, which is significantly larger than what is allowed by the $d = \mathcal{O}(\log n)$ criterion of Jurdziński and Lazić [JL17, Theorem 7]. Indeed, its rate of growth is much larger than any poly-logarithmic function of n, because for every positive constant c, we have $(\lg n)^c = 2^{c \cdot \lg \lg n}$, and $c \cdot \lg \lg n$ is exponentially smaller than $b \cdot \sqrt{\lg n}$. At the same time, the $\mathcal{O}\left(\sqrt{\log n}\right)$ rate of growth allowed in this scenario for the Strahler number k substantially exceeds $k = \mathcal{O}(1)$ required by Lehtinen [Leh18, Theorem 3.6].

Part II

Chapter 6

Strategy iteration algorithm with decompositions

Strategy improvement algorithms are a class of algorithms to solve Markov decision processes as well as parity, mean-payoff, discounted payoff and stochastic games. This approach was motivated by Howard's [How60] policy iteration algorithm, which is commonly used to determine the values of a Markov decision process. Hoffman and Karp [HK66] pioneered the development of strategy improvement algorithms for two-player games and addressed stochastic games. These algorithms trickled down the hierarchy of games from turn-based stochastic games [Con92] to discounted and mean-payoff games [GKK88, Pur95, ZP96], to parity games [VJ00].

Strategy improvement algorithms for positionally determined games fix a valuation for every Steven strategy, usually based on an optimal counter-strategy of Audrey. Starting from a positional strategy of Steven, until an optimal strategy is found, the strategy is *improved* with respect to this valuation by switching some edges of the strategy for Steven. Ideally, computing this improved strategy and verification of its optimality can be done efficiently. The proceedure that dictates which edges are chosen is called the *switching policy*.

Variations of strategy improvement algorithms have been explored for parity games, resulting in both theoretical and practical studies [VJ00, Lut08, Sch08, Fea17, FS18]. Despite the popular notion that the number of strategy improvements required is generally small in practice, these algorithms can have exponential worst-case complexity. Exponential families of examples for common pivoting rules for both games [Fri09] and for MDPs [Fea10] emerged decades after the introduction of strategy improvement algorithms, revealing that strategy improvement algorithms do not always guarantee polynomial termination as initially believed. Apart from

deterministic switching conditions, subexponential lower bounds were established even for randomised pivoting rules [FHZ11].

The quasi-polynomial breakthrough for parity games inspired further study of this classical algorithm. Ohlmann, in his PhD thesis [Ohl21], outlined an impossibility result that ruled out quasi-polynomial algorithms for a particular framework of strategy improvement algorithms. This framework required that the valuation was based on the current strategy and an optimal positional counter strategy. However, the work of Koh and Loho [KL22] circumvented this impossibility result by considering a hybrid algorithm of value iteration and strategy improvement (called *strategy iteration* algorithms here) where the valuation depended on both an Audrey strategy and also on a progress measure.

In the strategy iteration algorithm of Koh and Loho for parity games, an arbitrary (positional) strategy is chosen for one player, say Audrey. The valuation at each step is a progress measure of the game obtained by restricting Audrey to the chosen strategy edges. When a new strategy is chosen for the next iteration, a new valuation is computed based on the progress measure of the previous iteration and the new strategy. This valuation is obtained by finding the smallest progress measure larger than the one previously computed on the game restricted to the new strategy. The technically challenging part of their algorithm is to find this smallest progress measure once a new strategy is identified. Koh and Loho provide a way to find such a progress measure for a new strategy in time $\mathcal{O}(mn^2 \log n \log d)$ if the underlying tree is the Jurdziński-Lazić universal tree [JL17], $\mathcal{O}(mn^2 \log^3 n \log d)$ if the underlying tree is the Strahler universal trees [DJT20], and $\mathcal{O}(d(m+n \log n))$ for complete trees [Jur00]. Combined with the bounds on the sizes of the tree, this gives them a quasi-polynomial strategy iteration algorithm for Jurdziński-Lazić universal trees and Strahler universal trees, and a new exponential strategy improvement algorithm.

Our primary contribution is a strategy iteration algorithm that improves on Koh and Loho's work in the following way. We produce a *universal strategy iteration algorithm*, whose underlying tree can be any tree, and not just the trees mentioned above. In order to achieve such an algorithm, we define new objects called decompositions, which can be seen as a relaxation of attractor decompositions [DJL18, DJL19], endowed with a partial order among them. Our algorithm works by iteratively improving an underlying valuation. However this valuation is based on the strategy, as well as on the decompositions maintained after each improvement of the strategy. Secondly, we show that using decompositions instead of progress measures simplifies the algorithm of Koh and Loho. Indeed, Koh and

Loho's quasi-polynomial algorithms relied on the regularity obtained from recursive definitions of the universal trees, whereas we are able to remove such requirements and modify their algorithm to work for parity games with arbitrary trees of attractor decompositions. Thirdly, we show that each strategy improvement step can be done in time $\mathcal{O}(md \log n)$. Especially in the case where the trees are Jurdziński-Lazić universal trees or the Strahler universal trees, we improve the runtime of one iteration of the strategy iteration algorithm to closely match Koh and Loho's runtime for complete trees.

6.1 Attractor decompositions versus decompositions

Attractor decompositions are structural witnesses that shed light on the underlying structure of the game. Although obtained naturally as a byproduct of the classical recursive symmetric attractor computation algorithm of McNaughton[McN93] and Zielonka [Zie98], utilising attractor decompositions in other quasi-polynomial versions of recursive attactor based algorithms—or even constructing them as a part of the output like the ones in Lehtinen et al. [LPSW22] or the universal algorithm of Jurdziński and Morvan [JMT22]—is not straightforward. This is because these attractor decompositions, by virtue of being witnesses of winning in a parity game, are quite binary: a subgame either has an attractor decomposition, or it doesn't.

We introduce the concept of *decompositions*, a relaxation of attractor decompositions, and use it as an ingredient in building a valuation. Since a valuation needs to determine "how good" a strategy is, it requires an underlying order. These decompositions are defined with respect to a fixed tree and, using the order of the fixed tree, we define an order on the set of all valuations. Observe that this definition deviates slightly from our earlier definition of an attractor decomposition, where they were defined as hierarchical decompositions of winning sets for parity games. The attractor decompositions thus defined had ordered trees that correspond to them. In order to facilitate our definition of a decomposition, we first introduce a slightly altered, but conceptually equivalent, definition of an attractor decomposition, which is defined with respect to a fixed ordered tree.

Since the attractor decompositions used in the following section for Steven and Audrey are defined with respect to a tree, we wish to reason about these trees slightly differently for each player.

Even and odd levels of trees. We fix a tree $\mathcal{T}$ which is equitable, that is, every leaf has the same "depth". For such a tree $\mathcal{T}$, we say that the *even level* of a leaf

of this tree is 2. For nodes of the tree $\mathcal{T}$ that are not the leaf, its *even level* is exactly two more than the level of any of its children. The *odd level* of a tree $\mathcal{T}$ is defined similarly, except that the odd level of the leaf is instead 1. Note that the height of a tree is at most half of the even or the odd level. For a node η in the tree $\mathcal{T}$, we use Even-level (η) and Odd-level (η) to denote the even and odd levels of η in the tree, respectively. The even level of a tree is the even level of its root.

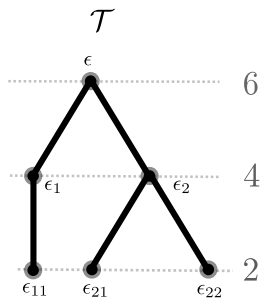


Figure 6.1: An ordered tree

For example, consider the labelled tree comprising exactly of the nodes $\{\langle\rangle, \langle 1\rangle, \langle 2\rangle, \langle 1,1\rangle, \langle 2,1\rangle, \langle 2,2\rangle\}$. This tree is illustrated in Fig. 6.1 where each node is represented by ϵ and its variants. The node $\langle 2,2\rangle$, written as ϵ_{22} has even level 2 and odd level 1 since it is a leaf, whereas the node $\langle\rangle$, written as ϵ, has even level 6 and odd level 5. For a node, we usually use the same variable, but with increasing subscripts from $\mathbb{N}$ to list their children in increasing order; for example, we use $\eta_1, \ldots, \eta_k$ to denote the first k children of η in that order.

In this chapter, we write $\mathcal{G}^{\leqslant p}$ to represent the subgame obtained by only considering the vertices of priority at most p in a parity game $\mathcal{G}$.

Attractor decomposition with respect to a tree $\mathcal{T}$. We alter our previous definition of an attractor decomposition. Our modified definition is more fundamentally associated with a pre-determined tree, instead of our earlier definition, where we had to extract the tree out of the recursive definition of the attractor decomposition. Hence, for a fixed tree, we define it as a partition of the set of vertices of the game such that there are three distinct parts of the partition corresponding to each node of the tree.

For a node η in a tree $\mathcal{T}$, we say that Steven has an $(\eta, \mathcal{T})$-*attractor decomposition* of a parity game $\mathcal{G}$, if the vertices of the game $\mathcal{G}$ can be partitioned into three times as many parts as there are descendants of the node η, with three parts corresponding to each of the nodes. We further require that these partitions satisfy some properties about traps and attractors.

More rigorously, consider an $(n, d{+}1)$-small parity game $\mathcal{G}$ where Steven wins from all vertices. Let η be a node in a tree $\mathcal{T}$, where Even-level $(\eta) \leqslant d$, and with ℓ children $\eta_1, \ldots, \eta_\ell$. We say that

$$\mathcal{A} = \left\langle H^\eta, T^\eta, (\mathcal{A}_1, \ldots, \mathcal{A}_\ell), S^\eta \right\rangle$$

is a Steven $(\eta, \mathcal{T})$-attractor decomposition of $\mathcal{G}$ if

1. the vertex set V is partitioned into $\ell + 3$ subsets by $H^\eta, T^\eta, S^\eta, R_1, \ldots, R_\ell$;
2. the set $V \setminus S^\eta$, henceforth written as $\left[\mathcal{A}^\eta\right]$, is a trap for Steven in the game $\mathcal{G}$ and it contains vertices of priority at most d;
3. H^η consists of all vertices in $\left[\mathcal{A}^\eta\right]$ that have priority exactly Even-level (η) ;
4. there is a Steven-reachability strategy from all vertices of T^η to the set H^η in the subgame induced by $\left[\mathcal{A}^\eta\right]$;
5. there is a Steven-reachability strategy from all vertices of S^η to the set of vertices $\left[\mathcal{A}^\eta\right]$ in $\mathcal{G}$;

and setting $\mathcal{G}_1 = \mathcal{G} \setminus \left(H^\eta \cup T^\eta\right)$, for $i = 1 \ldots \ell$, we have:

6. R_i is a trap for Steven in $\mathcal{G}_i$;
7. $\mathcal{A}_i$ is an $(\eta_i, \mathcal{T})$-attractor decomposition for the subgame induced by R_i;
8. $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus R_i$;

and $\mathcal{G}_{\ell+1} = S^\eta$.

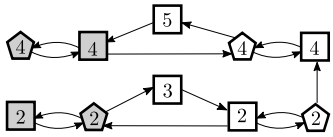
For any node γ that is a descendent of η, we write $H^\gamma_{\mathcal{A}}$, $T^\gamma_{\mathcal{A}}$ or $S^\gamma_{\mathcal{A}}$ to denote the sets H^γ, T^γ or S^γ respectively in an $(\eta, \mathcal{T})$-decomposition $\mathcal{A}$ of $\mathcal{G}$. The subscript is only adopted in situations where we refer to more than one attractor decomposition and we drop the subscript if $\mathcal{A}$ is clear from context.

Remark 3. *For a tree $\mathcal{T}$ and a node η in it with even level d, a Steven $(\eta, \mathcal{T})$-decomposition of an $(n, d+1)$-small parity game $\mathcal{G}$ always is such that*

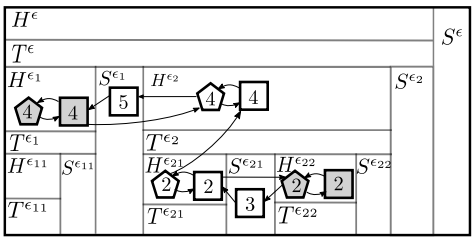
- *$H^\gamma_{\mathcal{A}}$ only contains vertices of priority exactly d,*
- *$T^\gamma_{\mathcal{A}}$ only contains vertices of priority at most $d-1$, and*
- *$S^\gamma_{\mathcal{A}}$ only contains vertices of priority at most $d+1$.*

An Audrey $(\eta, \mathcal{T})$-decomposition is defined analogously. We sometimes write $\mathcal{T}$-decomposition instead of $(\eta, \mathcal{T})$-decomposition when η is the root of $\mathcal{T}$.

Example 2. *Consider the game $\mathcal{G}$ illustrated in Fig. 6.2(a). Steven wins from all vertices in this game. We show a $\mathcal{T}$-attractor decomposition in Fig. 6.2(b) of the game $\mathcal{G}$, where the tree $\mathcal{T}$ is the $\mathbb{N}$-labelled tree illustrated in Fig. 6.1. In a Steven $(\epsilon, \mathcal{T})$-attractor decomposition, the sets $H^\epsilon, T^\epsilon, R_1, R_2, S^\epsilon$ as in the definition of an*



(a) A parity game $\mathcal{G}$



(b) $(\epsilon, \mathcal{T})$-attractor decomposition $\mathcal{A}$ of $\mathcal{G}$

Figure 6.2: A parity game and its attractor decomposition

attractor decomposition are as follows. The parts $H^{\epsilon}, T^{\epsilon}$, and S^{ϵ} are empty. The set R_1 contains all the parts to the left of and including the part S^{ϵ_1}. The set R_2 contains all other parts strictly to the right of and not including S^{ϵ_1}. A reader is encouraged to verify Remark 3 on this example.

The set S^{ϵ} is used to denote attractor sets represented on the "**S**ide" of the game, and T^{η} is used to denote the attractor sets that are attracted to "**T**op" of the game where the "**H**ighest priority" set H^{ϵ} is placed. We also often refer to the sets as top-attractor set or side-attractor set of a node γ for the sets T^{γ} and S^{γ} respectively.

Proposition 6.1.1 ([McN93, Zie98])**.** *In an (n, d)-small parity game $\mathcal{G}$, Steven wins from all vertices if and only if there is a Steven $(\eta, \mathcal{T})$-attractor decomposition for some node η in an ordered tree $\mathcal{T}$ where the even level of η is at most $d + 1$.*

Steven attractor decomposition when Audrey has no choice. Finding attractor decompositions of a parity game is at least as hard as identifying a winner of a parity game, as atrractor decompositions are a witness of winning. If we restrict ourselves to the case where Audrey has no choice (all her vertices have at most one out-going edge), solving such games can be done in near-linear time. This is not surprising, as the problem of finding if Steven can win reduces to finding an even cycle in the underlying graph. One can do this in time $\mathcal{O}(md)$ by using Tarjan's SCC [Tar72] decomposition algorithm on the graph restricted to vertices of priority at most p, for each even p, to identify such cycles. King, Kupferman, and Vardi [KKV01] improved the complexity by providing an $\mathcal{O}(m \log d)$ time algorithm for checking non-emptiness of a parity automaton by identifying the states of an automaton, from which some even-cycle can be reached. Finding the winner in a parity game where a player has no choice reduces to the problem of checking non-emptiness of a parity automaton.

In such games where Audrey has no choice, Steven has a "simple" attractor decomposition. By simple, we mean that Audrey has an $\mathcal{I}$-decomposition where $\mathcal{I}$ is the tree with exactly one leaf and the even level of its root is as large as the highest even priority in the game $\mathcal{G}$.

Proposition 6.1.2. *For an (n, d)-small parity game, where the root of the tree with one leaf $\mathcal{I}$ has even level d, there is a Steven $\mathcal{I}$-attractor decomposition of $\mathcal{G}$, if Audrey has no choice in $\mathcal{G}$.*

Proof. We assume that we have a parity game where an even cycle is reachable by any vertex. If not, one could apply the algorithm of King, Kupferman, and Vardi [KKV01] to remove vertices that cannot reach an even cycle. Let the root of $\mathcal{I}$ be η and the only child of η be η_1.

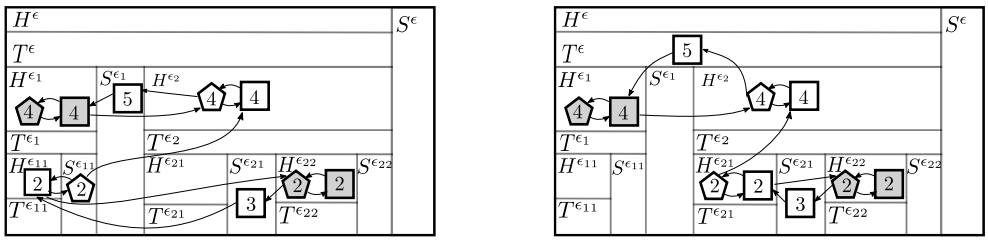
We define the $(\eta, \mathcal{I})$-decomposition $\mathcal{A}^\eta$ defined as follows:

- The set S is the set of vertices in $\mathcal{G}$ from which all paths lead to a vertex of priority $d + 1$.
- The set H is the set of vertices of priority d from which there is an infinite path that does not visit a vertex of priority $d + 1$.
- The set T is the set of vertices of priority at most $d - 1$ and from which there is a path to H in $\mathcal{G}^{\leqslant d}$.
- The set W is the set of vertices in $\mathcal{G}$ that are not in S, H, or T.
- Recursively, we find the $(\eta_1, \mathcal{I})$-attractor decomposition $\mathcal{A}'$ for the subset of vertices W in the rest of the game $\mathcal{G}^{\leqslant d-1}$, where the node η_1 is a child of the node η.
- Declare $\mathcal{A} = \left\langle H, T, \left(\mathcal{A}'\right), S \right\rangle$ as the $(\eta, \mathcal{I})$-attractor decomposition. □

Suppose we fix a strategy for Audrey, then it is easy according to Proposition 6.1.2 above to find an attractor decomposition. But what happens to this attractor decomposition when a different strategy of Audrey is chosen? To understand this further, we introduce our central object of this chapter: a decomposition. It is a relaxation of the above definition of a Steven $(\eta, \mathcal{T})$-attractor decomposition of a parity game.

Decomposition. A *Steven $(\eta, \mathcal{T})$-decomposition* $\mathcal{D}^\eta$ of the vertices V of a parity game $\mathcal{G}$, where children of η are denoted by $\eta_1, \ldots, \eta_k$, is defined recursively as

$$\mathcal{D}^\eta = \left\langle H^\eta, T^\eta, \left(\mathcal{D}^{\eta_1}, \ldots, \mathcal{D}^{\eta_k}\right), S^\eta \right\rangle$$



(a) Decomposition $\mathcal{D}$ (b) Decomposition $\mathcal{E}$

Figure 6.3: Two decomposition of the parity game in Fig. 6.2(a) neither of them is an attractor decomposition

such that

- the sets $H^\eta, T^\eta, S^\eta, R_1, \ldots,$ and R_k partition V.
- H^η consists exactly all vertices of priority equal to Even-level (η) in set $V \setminus S^\eta$, written as $\left[\mathcal{D}^\eta\right]$ henceforth.
- S^η is a superset of all vertices in V of priority Even-level $(\eta) + 1$ in V.
- For each child $\eta_1, \eta_2, \ldots, \eta_k$ of η in $\mathcal{T}$, $\mathcal{D}^{\eta_i}$ is an $(\eta_i, \mathcal{T})$-decomposition of R_i.

Observe that although there is no condition on the set T^η in the above definition, it only contains vertices of priority at most Even-level $((\,)\,\eta) - 1$. Given an $(\eta, \mathcal{T})$ decomposition, we refer to all the vertices of $\mathcal{G}$ using $[\![\mathcal{D}^\eta]\!]$. In the above, $[\![\mathcal{D}^\eta]\!] = V$ and $[\![\mathcal{D}^{\eta_i}]\!] = R_i$. Similar to attractor decompositions, when η is the root of $\mathcal{T}$, we just refer to it as $\mathcal{T}$-decomposition of a game. The Audrey $(\eta, \mathcal{T})$-decomposition of a game is defined analogously. We sometimes refer to a set of the form T^γ for a node γ as the top-set of γ, as it is derived from the top attractor set, and similarly we say side-set to refer to sets of the form S^γ.

Example 3. *Consider the two different decompositions given in Fig. 6.3 of the game $\mathcal{G}$ in Fig. 6.2(a). These decompositions are non-examples of attractor decomposition. Neither decomposition here is an attractor decomposition. The decomposition $\mathcal{D}$ in Fig. 6.3(a) is not an attractor decomposition because the set of vertices $[\![\mathcal{D}^{\epsilon_1}]\!]$ does not form a trap for Steven in the game. The set $[\![\mathcal{D}^{\epsilon_1}]\!]$ consisting of the shaded vertices of priority 4 and the non-shaded vertices of priority 2 does not form a trap for Steven as Steven can escape to $H^{\epsilon_2}_{\mathcal{D}}$. The decomposition $\mathcal{E}$ in Fig. 6.3(b) is not an attractor decomposition because $T^\epsilon_{\mathcal{E}}$ contains the vertex of priority 5 and this vertex is not in the Steven attractor to $H^\epsilon_{\mathcal{E}}$.*

6.2 Valuation using leafy trees

Our goal of introducing the concept of decompositions was to construct a valuation using it. To achieve this goal, we need these decompositions to have a (partial) ordering between them. In this regard we first define a modification of a given tree $\mathcal{T}$, called the *leafy tree of* $\mathcal{T}$. The nodes of these leafy trees also have a total order. We argue that a $\mathcal{T}$-decomposition uniquely defines a specific map from vertices into leafy tree of $\mathcal{T}$. Further, this map defines a partial order between decompositions which is inherited from the order on leafy trees.

Leafy trees. We define the *leafy tree* of a (labelled) tree $\mathcal{T}$ as the tree, which in addition to all nodes in the tree $\mathcal{T}$, contains two new children η^S and η^T for each node η in the tree $\mathcal{T}$ and a unique distinct element $\top$. These newly introduced nodes η^S and η^T are declared to be the smallest and the largest of the children of each node η and $\top$ treated as the last child of the root of $\mathcal{T}$. More formally, we say $\mathcal{L}(\mathcal{T})$ is the leafy tree of $\mathcal{T}$ where

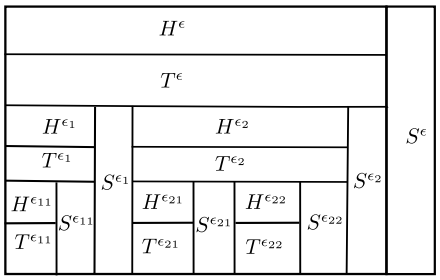
$$\mathcal{L}(\mathcal{T}) = \bigcup_{\eta \in \mathcal{T}} \left\{\eta, \eta^S, \eta^T\right\} \cup \{\top\}.$$

The order of elements in $\mathcal{L}(\mathcal{T})$ is inherited from the tree order on $\mathcal{T}$. The underlying tree order induces a total order on the set of all nodes of $\mathcal{L}(\mathcal{T})$ with $\top$ as the largest element.

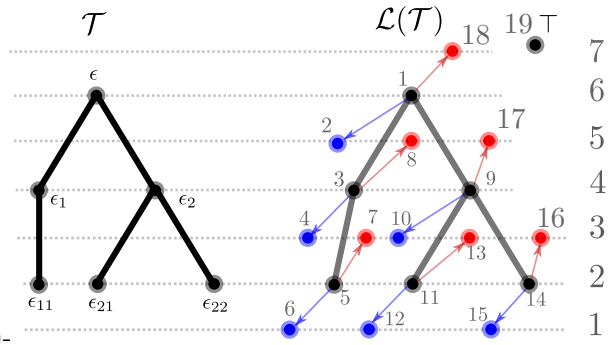
Observe that unlike the trees we had considered so far, leafy trees are not always equitable. Nonetheless, we extend some definition to leafy trees $\mathcal{L}(\mathcal{T})$ of an equitable tree $\mathcal{T}$. We say even (resp. odd) level of η^S to be one more than the even (odd) level of η and the even (odd) level of η^T to be one less. The even (resp. odd) level of $\top$ is one more than that of the even (odd) level of the root. Fig. 6.4(b) illustrates the leafy tree of a tree with three leaves, which can be seen as an $\mathbb{N}$-labelled tree whose nodes are $\{\langle\rangle, \langle 1\rangle, \langle 2\rangle, \langle 1,1\rangle, \langle 2,1\rangle, \langle 2,2\rangle\}$.

For an element x of $\mathcal{L}(\mathcal{T})$ and p, a number that is at most the even level of the tree, we define next (x, p) to be the smallest element in $\mathcal{L}(\mathcal{T})$ that is larger than x which has even level exactly p. For an element x in $\mathcal{L}(\mathcal{T})$ and an even value p, we define $x|_p$ (similar to Chapter 4) to be the ancestor of x at level p. If the current node x has a level larger than p, then we define it to be x.

A strategy respecting decomposition. Recall that a strategy of Audrey in a game $\mathcal{G}$ is a set of some edges outgoing from Audrey' vertices and all edges outgoing from Steven's vertices. We define the restriction of a game $\mathcal{G}$ to a positional strategy



(a) A representation of an $(\epsilon, \mathcal{T})$-decomposition of a game



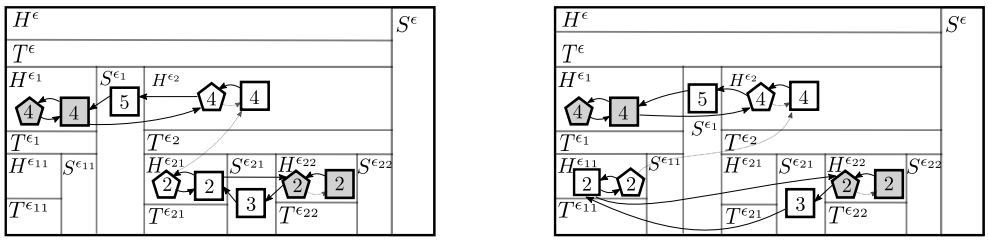
(b) The tree $\mathcal{T}$ and its leafy tree $\mathcal{L}(\mathcal{T})$ with level and order.

ρ as the same game instead on the arena (V, ρ) where V is the set of vertices of $\mathcal{G}$. We write $\mathcal{G}|_\rho$ to denote this game $\mathcal{G}$ restricted to a strategy ρ. Consider a Steven $(\eta, \mathcal{T})$-decomposition $\mathcal{D}$ of a parity game $\mathcal{G}$ and let ρ be a positional Audrey strategy in $\mathcal{G}$. We say that this decomposition $\mathcal{D}$ is a *ρ-respecting decomposition* of the game $\mathcal{G}$, if it is a Steven $(\eta, \mathcal{T})$-*attractor* decomposition of the (restricted) game $\mathcal{G}|_\rho$.

A valuation of a strategy and decomposition pair. Consider a Steven $(\eta, \mathcal{T})$-decomposition $\mathcal{D}$ that is ρ-respecting, and node γ of the tree $\mathcal{T}$, which is a descendant of node η. For each vertex $v \in H^\gamma_\mathcal{D} \cup S^\gamma_\mathcal{D} \cup T^\gamma_\mathcal{D}$, we determine two values: the first is a node in the leafy tree and the second is a natural number. In cases where $v \in H^\gamma_\mathcal{D}$, the second component is automatically declared to be 0. Otherwise, the second component corresponds to the the length of the shortest path in $\mathcal{G}_\rho$ from v to outside the set $S^\gamma_\mathcal{D}$ or $T^\gamma_\mathcal{D}$ that visits the set $\lceil \mathcal{D}^\eta \rceil$ or H^η, respectively (if there is no such path, then we declare this length as 1). The first component is γ, γ^S, or γ^T, depending on weather v belongs to $H^\gamma_\mathcal{D}$, $S^\gamma_\mathcal{D}$ or $T^\gamma_\mathcal{D}$, respectively.

For an Audrey strategy ρ and a Steven $\mathcal{T}$-decomposition $\mathcal{D}$ of an (n, d)-small parity game $\mathcal{G}$ that is ρ-respecting, we define the valuation of $\mathsf{val}(\mathcal{D}, \rho)$ to be a map from V to the set $\mathcal{M}(\mathcal{T})$ that we define. This set $\mathcal{M}(\mathcal{T})$ consists of tuples where the first element of the tuple is from $\mathcal{L}(\mathcal{T})$ and the second element is a natural number that is at most $n-1$. In other words, we would have for each $\eta \in \mathcal{T}$, elements of the form $(\eta, 0), \left(\eta^T, i\right)$, and $\left(\eta^S, i\right)$, for all i element of $\{1, \ldots, n-1\}$. We make an exception for the element $\top$ in the leafy tree $\mathcal{L}(\mathcal{T})$, where this is instead considered as a unique element in $\mathcal{M}(\mathcal{T})$ as well. This space of valuations $\mathcal{M}(\mathcal{T})$ is defined as

$$\mathcal{M}(\mathcal{T}) = \{\top\} \cup \bigcup_{\eta \in \mathcal{T}} \{(\eta, 0)\} \cup \bigcup_{\eta \in \mathcal{T}} \left\{\eta^T, \eta^S\right\} \times \{1, \ldots, n-1\}.$$



(a) Decomposition $\mathcal{A}$ (b) Decomposition $\mathcal{D}$

Figure 6.5: Two decomposition of the parity game $\mathcal{G}$ in Fig. 6.2(a) with its strategy edges highlighted

We define the valuation $\mathsf{val}\left(\mathcal{D}, \rho\right)\left(v\right)$ as

- $(\eta, 0)$ if $v \in H^{\eta}$;
- $\left(\eta^{T}, i\right)$ if $v \in T^{\eta}$ where i is the length of the shortest path in $\mathcal{G}|_{\rho}$ from v to the set H^{η};
- $\left(\eta^{S}, i\right)$ if $v \in S^{\eta}$ where i is the length of the shortest path in $\mathcal{G}|_{\rho}$ from v to the set $\left[\mathcal{D}^{\eta}\right]$;
- $\top$, for any v not in the set $[\![\mathcal{D}]\!]$.

The valuation $\mathsf{val}\left(\mathcal{D}, \rho\right)$ is defined only when $\mathcal{D}$ is a ρ-respecting decomposition.

An ordering of the valuations. The ordering $\preceq$ defined on $\mathcal{L}\left(\mathcal{T}\right)$ is also used to denote the ordering we had defined on $\mathcal{M}(\mathcal{T})$. For two elements η, η^{S}, the elements $(\eta, 0) \prec \left(\eta^{S}, 3\right)$ in $\mathcal{M}(\mathcal{T})$ since $\eta \prec \eta^{S}$ and $\left(\eta^{S}, 3\right) \prec \left(\eta^{S}, 5\right)$, since $3 < 5$.

This further naturally defines an ordering $\sqsubseteq$ for the set of all valuations by extending the $\preceq$ ordering on $\mathcal{M}(\mathcal{T})$ further to the point-wise ordering over functions from V to $\mathcal{M}(\mathcal{T})$.

For two strategies ρ and σ of Audrey and for two Steven $\mathcal{T}$-decompositions $\mathcal{D}$ and $\mathcal{E}$ of a game $\mathcal{G}$ such that $\mathcal{D}$ is ρ-respecting and $\mathcal{E}$ is σ-respecting, we can compare the decomposition-strategy pair $\mathcal{D}$ and ρ with the pair $\mathcal{E}$ and σ by comparing $\mathsf{val}\left(\mathcal{D}, \rho\right)$ and $\mathsf{val}\left(\mathcal{E}, \sigma\right)$. Moreover, we say $\mathsf{val}\left(\mathcal{D}, \rho\right) \sqsubsetneq \mathsf{val}\left(\mathcal{E}, \sigma\right)$ if $\mathsf{val}\left(\mathcal{D}, \rho\right) \sqsubseteq \mathsf{val}\left(\mathcal{E}, \sigma\right)$ and $\mathsf{val}\left(\mathcal{D}, \rho\right)\left(v\right) \neq \mathsf{val}\left(\mathcal{E}, \sigma\right)\left(v\right)$ for at least some v.

Example 4. *Consider the attractor decomposition $\mathcal{A}$ in Fig. 6.5(a) and the decomposition $\mathcal{D}$ in Fig. 6.5(a). Let the Audrey strategy ρ be the one which contains the edges between*

- *the shaded priority* 2 *Audrey vertex and the priority* 3 *Steven vertex,*

- *the unshaded priority* 2 *Audrey vertex and the adjacent priority* 2 *vertex,*
- *the unshaded priority* 4 *Audrey vertex and the priority* 5 *Steven vertex, and*
- *the shaded priority* 4 *Audrey vertex and the shaded priority* 4 *Steven vertex*

along with all of Steven's outgoing edges. The edges that are not in the strategy are dotted.

We then have $\mathsf{val}(\mathcal{D}, \rho) \sqsubseteq \mathsf{val}(\mathcal{A}, \rho)$*, since the decomposition on all but the unshaded vertices of priority* 2 *are the same. For vertices of priority* 2*, the valuation of* $\mathcal{D}$ *and* ρ *at these vertices are* $(\epsilon_{11}, 0)$*, whereas the valuation of* $\mathcal{A}$ *and* ρ *at these vertices are* $(\epsilon_{21}, 0)$ *for both, which is larger.*

In a scenario where $\mathsf{val}(\mathcal{D}, \rho) \not\sqsubseteq \mathsf{val}(\mathcal{E}, \sigma)$, we say that the valuation of the strategy and decomposition pair $\mathsf{val}(\mathcal{E}, \sigma)$ is *strictly improving* from $\mathsf{val}(\mathcal{D}, \rho)$. We argue that for a fixed strategy σ, there is a minimum decomposition $\mathcal{E}$ such that the valuation $\mathsf{val}(\mathcal{E}, \sigma)$ is strictly improving from the valuation $\mathsf{val}(\mathcal{D}, \rho)$. The lemma below shows that such a minimum decomposition exists. If the strategy σ is clear from context, we also just say the decomposition $\mathcal{E}$ is strictly improving, rather than the valuation $\mathsf{val}(\mathcal{E}, \sigma)$.

Lemma 6.2.1. *Given two* $(\eta, \mathcal{T})$*-decompositions* $\mathcal{D}$ *and* $\mathcal{E}$ *of an* (n, d)*-small parity game such that both are* σ*-respecting decompositions for a Steven strategy* σ*, there is a* σ*-respecting decomposition* $\mathcal{F}$ *such that* $\mathsf{val}(\mathcal{F}, \sigma) = \min\{\mathsf{val}(\mathcal{D}, \sigma), \mathsf{val}(\mathcal{E}, \sigma)\}$.

Proof. Consider the function f defined from the vertex set V to $\mathcal{M}(\mathcal{T})$ as $f(v) = \min\{\mathsf{val}(\mathcal{D}, \sigma)(v), \mathsf{val}(\mathcal{E}, \sigma)(v)\}$, for each vertex v. Now, we define $\mathcal{F}$ to be the decomposition such that $H^{\gamma}_{\mathcal{F}} = \{ v \mid f(v) = (\gamma, 0) \}$, $T^{\gamma}_{\mathcal{F}} = \{v \mid f(v) = \left(\gamma^T, i\right)$ for some $i\}$ and $S^{\gamma}_{\mathcal{F}} = \left\{ v \mid f(v) = \left(\gamma^S, i\right) \text{ for some } i \right\}$. We show that $\mathcal{F}$ is a σ-respecting decomposition and also that $\mathsf{val}(\mathcal{F}, \sigma) = f = \min\{\mathsf{val}(\mathcal{D}, \sigma), \mathsf{val}(\mathcal{E}, \sigma)\}$.

We show using an induction on the number of nodes in the tree. Our induction hypothesis is that for all v, if $f(v) = \min\{\mathsf{val}(\mathcal{D}, \sigma)(v), \mathsf{val}(\mathcal{E}, \sigma)(v)\} \geqslant (\eta, 0)$, then $\mathcal{F}$ so constructed is an $(\eta, \mathcal{T})$-decomposition. The base case is established by the simple observation that the induction hypothesis is true if η is the leaf of the tree $\mathcal{T}$. To show that $\mathcal{F}$ satisfies the conditions of an attractor decomposition on restricting $\mathcal{G}$ to σ. We list the properties as in the definition of an attractor decomposition (in order) and show that they are satisfied.

1. The corresponding sets form a partition follows from the fact that f is a map into the leafy tree.

2. $\left[\mathcal{F}^{\eta}\right]$ is a trap for Steven. Indeed, due to our assumption, for all vertices v, $f(v) \succeq (\eta, 0)$. Therefore, since $\left[\mathcal{F}^{\eta}\right]$ since it contains $\left[\mathcal{D}^{\eta}\right] \cup \left[\mathcal{E}^{\eta}\right]$, $\left[\mathcal{F}^{\eta}\right]$ is exactly $\left[\mathcal{D}^{\eta}\right] \cup \left[\mathcal{E}^{\eta}\right]$. The union of two traps for Steven is still a trap for him. $H^{\eta}_{\mathcal{F}}$ consists of vertices of priority equal to the even level of η. Moreover, note that all vertices v in $H^{\eta}_{\mathcal{D}}$ or $H^{\eta}_{\mathcal{E}}$ consists of the minimal elements for $\mathsf{val}(\mathcal{D}, \sigma)$ and $\mathsf{val}(\mathcal{E}, \sigma)$ respectively.

3. $H^{\eta}_{\mathcal{F}}$ only consists of vertices of priority at most d since both $H^{\eta}_{\mathcal{D}}$ and $H^{\eta}_{\mathcal{E}}$ only contains vertices of priority at most d.

4. There is a path from $T^{\eta}_{\mathcal{D}}$ and $T^{\eta}_{\mathcal{E}}$, which visits $H^{\eta}_{\mathcal{F}} = H^{\eta}_{\mathcal{D}} \cup H^{\eta}_{\mathcal{E}}$. $T^{\eta}_{\mathcal{F}}$ consists of $\left(T^{\eta}_{\mathcal{D}} \cup T^{\eta}_{\mathcal{E}}\right) \setminus H^{\eta}_{\mathcal{F}}$. Therefore, there is a Steven-reachability strategy from $T^{\eta}_{\mathcal{F}}$. This means that there is a path from each vertex in $T^{\eta}_{\mathcal{F}}$, which visits $H^{\eta}_{\mathcal{F}}$.

5. there is a Steven-reachability strategy from $S^{\eta}_{\mathcal{F}}$ to $\left[\mathcal{F}^{\eta}\right]$ in the subgame $\mathcal{G}$ and this can be shown by using arguments similar to those used in item 4.

Since η_1 is the smallest child of η after η^T, we know that for vertices v in $\mathcal{G}_1 = \mathcal{G} \setminus \left(H^{\eta}_{\mathcal{F}} \cup T^{\eta}_{\mathcal{F}}\right)$, we have $f(v) \succcurlyeq (\eta_1, 0)$. Henceforth, we assume for each $v \in \mathcal{G}_i$, we have $f(v) \succcurlyeq (\eta_i, 0)$. For $i = 1, \ldots \ell$, we can show that

6. if R_i is the set of vertices in $\mathcal{A}^{\eta_i}_{\mathcal{F}}$. then R_i forms a trap. This again follows from "additive" property of traps, similar arguments to arguing that $\left[\mathcal{F}^{\eta}\right]$ was a trap;

7. the decomposition $\mathcal{A}^{\eta_i}_{\mathcal{F}}$ is in fact an $(\eta_i, \mathcal{T})$-attractor decomposition of R_i due to our inductive assumption;

8. on setting $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus R_i$, we are now only left with vertices $v \in \mathcal{G}_{i+1}$ such that $f(v) \succeq (\eta_{i+1}, 0)$.

The fact that $\mathsf{val}(\mathcal{F}, \sigma) = \min\{\mathsf{val}(\mathcal{D}, \sigma), \mathsf{val}(\mathcal{E}, \sigma)\}$ follows from the definition of the decomposition $\mathcal{F}$. □

We say that a Steven decomposition $\mathcal{D}$ is a *minimum* ρ-respecting decomposition of $\mathcal{G}$ if the valuation $\mathsf{val}(\mathcal{D}, \rho)$ is the point-wise minimum among the set of all ρ-respecting decomposition. Given a ρ-respecting Steven decomposition $\mathcal{D}$ and a strategy σ such that $\mathcal{D}$ is not σ-respecting, there is a unique $\mathcal{E}$, which is σ-respecting and also satisfies two properties: (1) $\mathsf{val}(\mathcal{E}, \sigma)$ is strictly improving from $\mathsf{val}(\mathcal{D}, \rho)$ and (2) $\mathsf{val}(\mathcal{E}, \sigma)$ is the minimum among the set of all σ-respecting decompositions $\mathcal{F}$ whose valuation $\mathsf{val}(\mathcal{F}, \sigma)$ is strictly improving from $\mathsf{val}(\mathcal{D}, \rho)$. The uniqueness follows from Lemma 6.2.1. We therefore define a decomposition $\mathcal{E}$ as *minimally*

σ-improving from $\mathsf{val}(\mathcal{D}, \rho)$ if the valuation $\mathsf{val}(\mathcal{E}, \sigma)$ is the minimum among all σ-respecting decompositions $\mathcal{F}$ of $\mathcal{G}$ such that $\mathsf{val}(\mathcal{F}, \sigma)$ is strictly improving from $\mathsf{val}(\mathcal{D}, \rho)$, in other words, it satisfies the above two conditions.

6.3 Strategy iteration with decompositions

Having defined the valuation for a strategy and a decomposition, we turn our attention to describing the strategy iteration algorithm based on this valuation. We give a high-level view of this algorithm, which is later described more formally in Algorithm 2. The algorithm takes as input an (n, d)-small parity game $\mathcal{G}$ and has access to a tree $\mathcal{T}$ whose root η has even level d. It starts with an Audrey strategy ρ, chosen using an arbitrary policy. Following this, a minimum ρ-respecting $(\eta, \mathcal{T})$-decomposition $\mathcal{D}$ of $\mathcal{G}$ is computed. Since the decomposition must be ρ-respecting, it is an attractor decomposition of the restricted game $\mathcal{G}|_\rho$. This can be done using methods proposed by Proposition 6.1.2, as any game where Audrey has no choice has a "simple" attractor decomposition.

At each iteration, with the help of a ρ-respecting decomposition $\mathcal{D}$, we pick a new Audrey strategy σ, using the procedure Improved-Strategy. For such Audrey strategies σ and ρ using decomposition $\mathcal{D}$, we find the minimally σ-improving decomposition $\mathcal{E}$ from the valuation $\mathsf{val}(\mathcal{D}, \rho)$. The iterative process restarts, but now with the strategy σ and the newly computed σ-respecting decomposition $\mathcal{E}$ instead. The algorithm terminates when the decomposition found is an attractor decomposition of the input game, or equivalently, the decomposition computed at the end of an iteration is σ-respecting for every Audrey strategy σ.

Algorithm 2 Decomposition based strategy iteration algorithm

Input: A game $\mathcal{G}$. ▷ The algorithm has access to the tree $\mathcal{T}$
Output: An attractor decomposition $\mathcal{D}$ of $\mathcal{G}$.

```
1: procedure Strategy Iteration(G)
2:     ρ ← an arbitrary strategy of Audrey.
3:     D ← minimum ρ-respecting decomposition of G
4:     repeat
5:         σ ← Improved-Strategy(G, D, ρ)
6:         E ← Minimal-Improve(G|_σ, val(D, ρ), η)
7:         ρ ← σ
8:         D ← E
9:     until D is an attractor decomposition of G
10:    return D
11: end procedure
```

There are two subroutines of the algorithm above that need to be discussed. Firstly, we describe an algorithm that picks the next strategy σ in the procedure Improved-Strategy (Algorithm 3). Secondly, we need to find a decomposition $\mathcal{E}$ that is minimally σ-improving from $\mathsf{val}(\mathcal{D}, \rho)$. This is described later in procedure Minimal-Improve (Algorithm 4).

Violating edges. To understand the procedure Improved-Strategy, which picks the strategy σ, we need the definition of a violating edge. Intuitively, an edge is violating with respect to a valuation if, along this edge, the valuation is 'non-decreasing'. An edge $u \to v$ of the game $\mathcal{G}$ with a $\mathcal{T}$-decomposition is said to be *violating* with respect to a valuation $\mathsf{val}(\mathcal{D}, \rho)$ if

- $u \in S^\eta$ or T^η and $\mathsf{val}(\mathcal{D}, \rho)(v) \succeq \mathsf{val}(\mathcal{D}, \rho)(u)$, or
- $u \in H^\eta$ and $\mathsf{val}(\mathcal{D}, \rho)(v) = (x, i)$ where $x \succeq \eta^S$.

Observe that no edge in ρ can be violating with respect to $\mathsf{val}(\mathcal{D}, \rho)$, since $\mathcal{D}$ is ρ-respecting. The procedure Improved-Strategy picks an Audrey strategy σ by swapping some (at least one) strategy edges in ρ for edges that are violating with respect to $\mathsf{val}(\mathcal{D}, \rho)$ as described below.

Algorithm 3 Returns a strategy σ that increases valuation in the next step

Input: A game $\mathcal{G}$, a decomposition $\mathcal{D}$, and a (Steven) strategy ρ.
Output: A (Steven) strategy σ of $\mathcal{G}$.

1: **procedure** Improved-Strategy($\mathcal{G}, \mathcal{D}, \rho$)
2: $E_{\mathsf{vio}} \leftarrow \{(u, v) \mid u$ is an Audrey vertex and (u, v) is violating in $\mathsf{val}(\mathcal{D}, \rho)\}$
3: σ is an Audrey strategy with at least one edge from E_{vio} and rest from ρ. ▷ The switching policy can be selected based on an algorithm, choosing to include a strict subset of all violating edges.
4: **return** σ
5: **end procedure**

The main technical challenge once an Audrey strategy σ is selected is in finding a minimum σ-improving decomposition from valuation $\mathsf{val}(\mathcal{D}, \rho)$. We describe an algorithm to find such a decomposition $\mathcal{E}$ with the help of the following definitions.

The anchor of cycle-winners. For a parity game where Audrey has no choice, we call cycle-winners for Steven the subset of vertices of the game that are even and have the highest priority in some (simple) cycle. Computing the set of cycle-winners can be done in time $\mathcal{O}(md)$. In the work of Koh and Loho [KL22], cycle-

winners at each step are fixpoints while performing a hybrid of progress-measure lifting and strategy improvement algorithms.

For constructing the $\mathcal{T}$-decomposition $\mathcal{E}$ that is the minimally σ-improving decomposition from $\mathsf{val}(\mathcal{D}, \rho)$, we also make use of such cycle-winners of the restricted game $\mathcal{G}|_\sigma$. As a first step to compute this decomposition $\mathcal{E}$, we define a $(\sigma, \mathcal{D})$-anchor denoted by $\mathfrak{L}^\sigma_\mathcal{D}$, which is a map from the set of cycle-winners of the game $\mathcal{G}|_\sigma$ to the nodes of the leafy tree $\mathcal{L}(\mathcal{T})$ that are also nodes in the tree $\mathcal{T}$. Intuitively, it maps all the cycle-winners of $\mathcal{G}$ to the smallest skeleton node whose even level is equal to its priority and is at least as large as its current valuation $\mathsf{val}(\mathcal{D}, \rho)$ when restricted to its first component. For a cycle-winner v in $\mathcal{G}|_\sigma$, we define the $(\sigma, \mathcal{D})$-anchor as

$$\mathfrak{L}^\sigma_\mathcal{D}(v) = \begin{cases} \eta & \text{if } v \in H^\eta \\ \text{next}\left(\eta^S, \pi(v)\right) & \text{if } v \in S^\eta \\ \text{next}\left(\eta^T, \pi(v)\right) & \text{if } v \in T^\eta. \end{cases}$$

Since the first component of $\mathsf{val}(\mathcal{D}, \rho)(v)$ is at least as large as $\mathfrak{L}^\sigma_\mathcal{D}(v)$, it is an "upper bound" for the first component of $\mathsf{val}(\mathcal{D}, \rho)(v)$ restricted to cycle-winners. We remark here that this map also turns out to be such an "upper bound" for the first component of the valuation $\mathsf{val}(\mathcal{E}, \sigma)$.

A description of Minimal-Improve. The procedure Minimal-Improve on a game $\mathcal{G}$, identifies the set R of all cycle-winners such that η is an ancestor of their $(\sigma, \mathcal{D})$-anchor. Three different sets with respect to R are computed.

1. the set S of vertices with a path to R, but all paths that reach R visit a vertex with the highest odd priority $d + 1$;
2. the set H of vertices of priority d that can reach a vertex in R, but without seeing any vertex of priority larger than d; and
3. the set T of vertices with a path to H, but which encounter vertices of priority at most $d - 1$ along the path.

On removing these three sets S, H, and T, the same process is repeated for each child η_i of node η in the tree, returning the respective $(\eta_i, \mathcal{T})$-decompositions. The algorithm finally returns the decomposition $\mathcal{E}$ obtained as $\langle H, T, \langle \mathcal{E}_1, \ldots, \mathcal{E}_k \rangle, S \rangle$. Vertices that cannot reach a cycle-winner are losing for Steven not only in the restricted game $\mathcal{G}|_\sigma$, but also in the larger game $\mathcal{G}$. This is because Audrey has a

strategy σ from each of these vertices, that ensures that Steven cannot win from them. We assume that while computing the set of cycle-winners, these vertices are removed by the algorithm and returned as winning for Audrey.

Algorithm 4 Returns the smallest minimally σ-improving decomposition $\mathcal{E}$

Input: (n, d)-small parity game $\mathcal{G}$ obtained by restricting game to Audrey strategy σ, a decomposition $\mathcal{D}$ of $\mathcal{G}$, and node of even level of η in the underlying tree $\mathcal{T}$.
Output: A subset of vertices W and its decomposition $\mathcal{E}$.

```
1: procedure MINIMAL-IMPROVE(𝒢, 𝒟, η)
2:     R ← {v | η is an ancestor of 𝔏^σ_𝒟(v)}
3:     H ← all priority d vertices with a path in 𝒢^{⩽d} to R
4:     T ← all vertices of priority ⩽ d − 1 with a path in 𝒢^{⩽d} to H
5:     W ← H ∪ T
6:     𝒢_1 ← 𝒢 \ W
7:     for each child η_1, ..., η_k of η in order do
8:         (W_i, ℰ_i) ←MINIMAL-IMPROVE(𝒢_i^{⩽d−1}, 𝒟, η_i)
9:         𝒢_{i+1} ← 𝒢_i \ W_i
10:        W ← W ∪ W_i
11:    end for
12:    S ← vertices for which all paths to W visit a priority d + 1 vertex
13:    W ← W ∪ S
14:    return (W, ℰ = ⟨H, T, (ℰ_1, ..., ℰ_k), S⟩)
15: end procedure
```

6.4 Correctness and running time of the algorithm

Our key contribution can be summarised by the following theorem.

Theorem D. *For a parity game $\mathcal{G}$ with n vertices, d priorities, and a tree $\mathcal{T}$ of even level d, each iteration of the* strategy iteration *algorithm (Algorithm 2 on page 91) takes time $\widetilde{\mathcal{O}}(|\mathcal{G}|d)$. The valuation (of the decomposition and strategy maintained) at each step is strictly improving. The algorithm terminates with a $\mathcal{T}$-attractor decomposition of $\mathcal{G}$ within $n^2|\mathcal{T}|$ iterations.*

We provide lemmas and propositions that form a building block for the proof of correctness and runtime of our algorithm. Computing the runtime of this algorithm is easily tackled with the following propositions.

Proposition 6.4.1. *For an (n, d)-small parity game $\mathcal{G}$ and a decomposition $\mathcal{D}$ and an Audrey strategy σ, computing $\mathfrak{L}^\sigma_\mathcal{D}$ takes time $\mathcal{O}(md)$, where m is the number of edges in $\mathcal{G}$.*

We assume that the procedure Minimal-Improve computes the set of cycle-winners as well as the $(\sigma, \mathcal{D})$-anchor $\mathfrak{L}^{\sigma}_{\mathcal{D}}$ in advance as it takes time $\mathcal{O}(md)$.

Lemma 6.4.2. *The procedure* Minimal-Improve *on an (n, d)-small game returns an output decomposition in time at most $\mathcal{O}(md \log n)$*

Proof. We assume the set of all cycle winners are computed and also arranged in ascending order of the value of $\mathfrak{L}^{\sigma}_{\mathcal{D}}$. Comparing each value takes $\mathcal{O}(d)$ time and therefore sorting takes at most time $\mathcal{O}(md \log n)$ and does not increase the claimed asymptotic runtime.

The algorithm's most crucial part outside the recursive call is identifying the set R and later, once this set of vertices R is identified, computing H, T and S. A naive analysis would indeed give us an unpleasant run-time of $\mathcal{O}(mnd)$. There are nd many possible values for η, and each η requires us to compute R, and its associated sets H, T and S, which would take $\mathcal{O}(m)$ time each. But, we show an amortised runtime of our algorithm is $\mathcal{O}(md \log n)$.

Once the cycle-winners are arranged in ascending order of their $\mathfrak{L}^{\sigma}_{\mathcal{D}}$ values, finding R for each sub-call on the node η amounts to doing a binary search for the interval which contains the values where η is an ancestor of $\mathfrak{L}^{\sigma}_{\mathcal{D}}$. This takes $d \log n$ time, since comparison takes time d and the binary search part takes $\mathcal{O}(\log n)$ time.

The crux of our argument reduces to showing that each edge is touched at most $\mathcal{O}(d)$ times overall outside of the operations discussed above. For this, we first make the following observations. Consider a vertex u in the game. It can reach some cycle-winner as the game is winning from every vertex. Suppose v is the cycle-winner that has the smallest anchor value $\mathfrak{L}^{\sigma}_{\mathcal{D}}(v)$ among all cycle-winners that u can reach. Let η be this value $\mathfrak{L}^{\sigma}_{\mathcal{D}}(v)$. In the decomposition $\mathcal{E}$, we have that $u \in H^{\gamma}_{\mathcal{E}}$ or $T^{\gamma}_{\mathcal{E}}$ or $S^{\gamma}_{\mathcal{E}}$ where $\gamma = \eta|_p$ for some even p. Such a value p is the largest value such that (1) there is a path in $\mathcal{G}^{\leq p}$ from u to v, or (2) there is a path in $\mathcal{G}^{\leq p+1}$. This is because by assumption, there is no path from u to any cycle-winner with a smaller value with respect to $\mathfrak{L}^{\sigma}_{\mathcal{D}}$.

We assume computing the reachability set is done using a backwards manner, that is, all the predecessors are iteratively added in a queue and the traversal is done using a BFS algorithm but for the revered graph $\mathcal{G}$. This is reminiscent of attractor computing algorithms for a game, that start from a target set and considers edges in reverse order. In this case, each vertex with an edge from vertex u is considered in either reachability set for at most d calls, one for each ancestor of the smallest cycle-winner that u can reach. Once it is identified as being in any of the sets $H^{\gamma}_{\mathcal{E}}$, $T^{\gamma}_{\mathcal{E}}$ or $S^{\gamma}_{\mathcal{E}}$, it is removed from the current set of vertices. This ensures that each

edge with u as a source is touched at most $\mathcal{O}(d\delta)$ times, where δ is the outdegree of u. The same holds for any vertex u, thus bounding the ammortised runtime by $\mathcal{O}(md \log n)$. □

Our most technical lemma is stated below, but we postpone the proof to later in the section.

Lemma 6.4.3. *Algorithm 2 on input an (n, d)-small parity game $\mathcal{G}$ and a tree $\mathcal{T}$ of even level at least d, satisfies the mono-variant that the the valuation $\mathsf{val}(\mathcal{D}, \rho)$ obtained from the Steven $\mathcal{T}$-decomposition $\mathcal{D}$ of $\mathcal{G}$ in Line 8 and Audrey strategy ρ in Line 7 is strictly improving from tuple formed by the valuation of the decomposition and strategy in the previous iteration.*

Assuming Lemma 6.4.3, and writing $\mathcal{D}_k$ and ρ_k to denote the values of $\mathcal{D}$ and ρ at the end of the k^{th} iteration, we get $\mathsf{val}(\mathcal{D}_1, \rho_1) \lneq \mathsf{val}(\mathcal{D}_2, \rho_2) \lneq \cdots \lneq \mathsf{val}(\mathcal{D}_k, \rho_k)$. There can be at most $n(2n-1)|\mathcal{T}|$ many functions from $V \to \mathcal{M}(\mathcal{T})$, the number of iterations cannot be larger than $n(2n-1)|\mathcal{T}|$. The algorithm terminates when $\mathcal{D}$ is an attractor decomposition. Since each step takes $\mathcal{O}(md \log n)$ according to Lemma 6.4.5 time our algorithm also takes time $\widetilde{O}(md(2n-1)|\mathcal{T}|)$.

For all Audrey strategies ρ, an attractor decomposition $\mathcal{A}$ is ρ-respecting. We can show inductively that for the k^{th} iteration, $\mathsf{val}(\mathcal{D}_k, \rho_k) \sqsubseteq \mathsf{val}(\mathcal{A}, \rho_k)$. Since we start from the smallest possible decomposition, and increase minimally at each step, we reach the attractor decomposition with the smallest valuation for all strategies ρ.

Tight, wobbly and stretched. Given a ρ-respecting decomposition $\mathcal{D}$ of a game $\mathcal{G}$, an edge $u \to v$ of a game $\mathcal{G}$ is said to be *tight* with respect to $\mathsf{val}(\mathcal{D}, \rho)$ if

- $\mathsf{val}(\mathcal{D}, \rho)(u) = \left(\eta^S, i\right)$ and $\mathsf{val}(\mathcal{D}, \rho)(v) = \left(\eta^S, i-1\right)$ for $i \in [1, n-1]$ and we say $\left(\eta^S, 0\right) \equiv (\eta, 0)$; or
- $\mathsf{val}(\mathcal{D}, \rho)(u) = \left(\eta^T, i\right)$ and $\mathsf{val}(\mathcal{D}, \rho)(v) = \left(\eta^T, i-1\right)$ for $i \in [1, n-1]$ and we say $\left(\eta^T, 0\right) \equiv (\eta, 0)$; or
- $\mathsf{val}(\mathcal{D}, \rho)(u) = (\eta, 0)$ and $\mathsf{val}(\mathcal{D}, \rho)(v) = (x, i)$ such that $x|_{\pi(u)} = \eta$.

Intuitively, an edge $u \to v$ is tight with respect to a valuation if the valuation of u was any smaller, the edge would be violating.

If there is an edge in a decomposition $\mathsf{val}(\mathcal{D}, \rho)$ that is neither tight nor violating, we call the edge *wobbly*. We call an Audrey positional strategy σ *stretched* with respect to $\mathsf{val}(\mathcal{D}, \rho)$ if the strategy σ (which consists of all of Steven's edges along with one edge from each of Audrey's vertices) has no wobbly edges.

Proposition 6.4.4. *Given a game $\mathcal{G}$, an Audrey strategy ρ and $\mathcal{D}$, a ρ-respecting decomposition, then* Improved-Strategy$(\mathcal{G}, \mathcal{D}, \rho)$ *returns a strategy σ that is stretched with respect to* $\mathsf{val}(\mathcal{D}, \rho)$.

Remark 4. *Any edge $u \to v$ in $\mathcal{G}|_\sigma$ where σ is stretched with respect to* $\mathsf{val}(\mathcal{D}, \rho)$ *is such that*

- $\mathsf{val}(\mathcal{D}, \rho)(u) = \left(\eta^S, k\right)$ *and* $\mathsf{val}(\mathcal{D}, \rho)(v) \geqslant \left(\eta^S, k-1\right)$ *or*
- $\mathsf{val}(\mathcal{D}, \rho)(u) = \left(\eta^T, k\right)$ *and* $\mathsf{val}(\mathcal{D}, \rho)(v) \geqslant \left(\eta^T, k-1\right)$ *or*
- $\mathsf{val}(\mathcal{D}, \rho)(u) = (\eta, 0)$ *and* $\mathsf{val}(\mathcal{D}, \rho)(v) = (x, i)$ *such that* $x|_{\pi(u)} \geqslant \eta$.

So far, we have observed that Improved-Strategy returns a stretched decomposition. Our next key lemma states that for a stretched decomposition $\mathcal{D}$ with respect to σ, the procedure Minimal-Improve$(\mathcal{G}, \mathcal{T}, \sigma, \mathcal{D})$ returns the respective minimum σ-improving decomposition from the current strategies.

Lemma 6.4.5. *Given a game $\mathcal{G}$, a ρ-respecting Steven decomposition $\mathcal{D}$, and an Audrey strategy σ that is a stretched with respect to the valuation* $\mathsf{val}(\mathcal{D}, \rho)$, *the procedure* Minimal-Improve$(\mathcal{G}, \mathcal{T}, \sigma, \mathcal{D})$ *returns the minimum σ-improving decomposition $\mathcal{E}$ from* $\mathsf{val}(\mathcal{D}, \rho)$.

To prove the lemma, we break it down further into two propositions, Propositions 6.4.6 and 6.4.7 which lead to the proof of lemma 6.4.5.

Henceforth we assume these lemmas are stated for a game $\mathcal{G}$, a strategy ρ of Audrey, a ρ-respecting decomposition $\mathcal{D}$, and a strategy σ that is stretched with respect to $\mathsf{val}(\mathcal{D}, \rho)$. Let $\mathcal{A}$ be the minimum σ-respecting decomposition with respect to $\mathsf{val}(\mathcal{D}, \rho)$.

Proposition 6.4.6 states that the anchor of a cycle winner helps find an upper bound on the value of the valuation of such a minimally σ-improving decomposition $\mathcal{A}$ in cycles. Later, Proposition 6.4.7 extends this to not just cycles but to vertices with paths to such cycles.

Proposition 6.4.6. *For a even cycle C in $\mathcal{G}|_\sigma$, let $\eta = \mathfrak{U}^\sigma_{\mathcal{D}}(v)$, then*

- $\mathsf{val}(\mathcal{D}, \rho)(v) \leqslant \mathsf{val}(\mathcal{A}, \sigma)(v) \leqslant (\eta, 0)$, *where v is a cycle-winner in the cycle C, and*
- $\mathsf{val}(\mathcal{D}, \rho)(u) \leqslant \mathsf{val}(\mathcal{A}, \sigma)(u) \leqslant \left(\eta^T, k\right)$, *for any u in the cycle C, and where k is the length of the path from u to some vertex of priority d in C.*

Proof. The first half of both of the inequalities that $\mathsf{val}(\mathcal{D},\rho)(v) \leqslant \mathsf{val}(\mathcal{A},\sigma)(v)$ and $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant \mathsf{val}(\mathcal{A},\sigma)(u)$, follow from $\mathcal{A}$ being the minimum σ-respecting decomposition with respect to $\mathsf{val}(\mathcal{D},\rho)$.

To show $\mathsf{val}(\mathcal{A},\sigma)(v) \leqslant (\mathfrak{Up}^{\sigma}_{\mathcal{A}}(v), 0)$, we construct a σ-respecting decomposition $\mathcal{B}$ such that (1) $\mathsf{val}(\mathcal{D},\rho) \sqsubseteq \mathsf{val}(\mathcal{B},\sigma)$ and (2) $\mathsf{val}(\mathcal{B},\sigma)(v) = (\mathfrak{Up}^{\sigma}_{\mathcal{A}}(v), 0)$. Since the valuation of $\mathcal{A}$ is smaller than any $\mathcal{B}$ that is σ-respecting, the inequality follows.

Since v is a cycle-winner, there is a cycle C where v has the highest even priority d in the cycle. Let the cycle C consists of vertices $u_0, \ldots, u_\ell$ and the edges in the cycle are from each u_i and u_{i+1}. Moreover, we refer to v as both u_0 and $u_{\ell+1}$ in our proof.

We define a σ-respecting decomposition $\mathcal{B}$ by defining two sets corresponding to the partitions of the decomposition as follows: $H^{\eta}_{\mathcal{B}} = \{u \mid u \text{ is a cycle-winner of } C\}$ and $T^{\eta}_{\mathcal{B}} = \{u \in C \mid u \text{ is not a cycle-winner of } C\}$. All the other sets in the decomposition are declared to be empty. Observe now that the valuation of such a $\mathcal{B}$ above along with the strategy σ can be deduced to be

$$\mathsf{val}(\mathcal{B},\sigma)(u) := \begin{cases} (\eta, 0) & u \text{ is a cycle-winner of } C \\ \left(\eta^T, k\right) & u \text{ has distance } k \text{ to the set of all cycle-winner in } \mathcal{G}_\sigma \\ \top & u \notin C. \end{cases}$$

We remark that in the cases where u is not a cycle-winner, but can reach one, k denotes the length of shortest path in $\mathcal{G}_\sigma$ to a cycle-winner. To show $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant \mathsf{val}(\mathcal{B},\sigma)(u)$, first observe for $u \notin C$, these vertices aren't present in the decomposition and hence its value is set to $\top$. We show $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant \mathsf{val}(\mathcal{B},\sigma)(u)$ for $u \in C$. We prove this using induction along with the fact that $\mathcal{D}$ is a stretched decomposition.

Base Case. For vertices u in the cycle whose shortest path to a cycle-winner of C is 0, we know $\mathsf{val}(\mathcal{B},\sigma)(u) \leqslant (\eta, 0)$ by definition. Let us assume $\left(\eta^T, 0\right)$ to also represent $(\eta, 0)$ henceforth.

Induction step. If for each vertex w in the cycle C, if the shortest path from w to a cycle-winner of C is i and $\mathsf{val}(\mathcal{D},\rho)(w) \leqslant \left(\eta^T, i\right)$, then for for vertices u in the cycle whose shortest path to a cycle-winner of C is $i+1$, from Remark 4, $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant \left(\eta^T, i+1\right)$.

Since σ-respecting decomposition labellings are closed under point-wise minimum from Lemma 6.2.1, the minimal σ-respecting decomposition $\mathcal{A}$ larger than

$\mathsf{val}(\mathcal{D},\rho)$ should also be such that $\mathsf{val}(\mathcal{D},\rho)(v) \leqslant \mathsf{val}(\mathcal{A},\sigma)(v) \leqslant \mathsf{val}(\mathcal{B},\sigma)(v)$. □

Proposition 6.4.7. *For a cycle-winner v of $\mathcal{G}_\sigma$ that has priority p and for some vertex u is such that (1) u has a greater even priority than p, that is, $\pi(u) = d \geqslant p$, where d is even and (2) there is a path from u to v in $\mathcal{G}|_\sigma^{\leqslant d}$, then*

- $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant (\eta, 0)$*; where* $\eta = \Updownarrow_{\mathcal{D}}^{\sigma}(v)|_d$.

Moreover, $\mathsf{val}(\mathcal{D},\rho)(w) \leqslant \left(\eta^T, k\right)$*, for any w that has a k-length path to u.*

Proof. Consider a path from u to v in $(\mathcal{G}|_\sigma)^{\leqslant d}$: $u \to u_1 \to u_2 \to \cdots \to u_k = v$. Since σ is stretched with respect to $\mathsf{val}(\mathcal{D},\rho)$, from Remark 4, we know that if we let the first component of $\mathsf{val}(\mathcal{D},\rho)(u_i)$ be x_i then we have $x_1|_d \leqslant \ldots x_i|_d \leqslant x_{i+1}|_d \leqslant \ldots \leqslant x_k|_d \leqslant \Updownarrow_{\mathcal{D}}^{\sigma}(v)|_d = \eta$. This shows $\mathsf{val}(\mathcal{D},\rho)(u) \leqslant (\eta, 0)$.

To show $\mathsf{val}(\mathcal{D},\rho)(w) \leqslant \left(\eta^T, k\right)$ is very close to the inductive proof in Proposition 6.4.6. This is done by the induction on the length of the shortest path to u.

□

Using Propositions 6.4.6 and 6.4.7 as building blocks, we prove Lemma 6.4.5. We restate the lemma again for convenience below.

Lemma 6.4.5. *Given a game $\mathcal{G}$, a ρ-respecting Steven decomposition $\mathcal{D}$, and an Audrey strategy σ that is a stretched with respect to the valuation $\mathsf{val}(\mathcal{D},\rho)$, the procedure* Minimal-Improve$(\mathcal{G},\mathcal{T},\sigma,\mathcal{D})$ *returns the minimum σ-improving decomposition $\mathcal{E}$ from $\mathsf{val}(\mathcal{D},\rho)$.*

Proof. Let $\mathcal{A}$ be the minimal σ-respecting decomposition with respect to $\mathsf{val}(\mathcal{D},\rho)$. We argue that $\mathsf{val}(\mathcal{E},\sigma) = \mathsf{val}(\mathcal{A},\sigma)$ where $\mathcal{E}$ is constructed by the procedure Minimal-Improve$(\mathcal{G}_\sigma,\eta,\Updownarrow_{\mathcal{D}}^{\sigma})$.

Induction hypothesis. If for each vertex v, $\mathsf{val}(\mathcal{D},\rho)(v) \geqslant (\eta,0)$ then Minimal-Improve$(\mathcal{G}|_\sigma,\mathcal{D},\mathcal{T},\eta_i)$ returns $\mathcal{E}$ and W such that

(a) $W = \left\{w \mid \mathsf{val}(\mathcal{A},\sigma)(w)| \leqslant \left(\eta^S, n-1\right)\right\}$.

(b) $\mathsf{val}(\mathcal{E},\sigma)(w) = \mathsf{val}(\mathcal{A},\sigma)(w)$ for all $w \in W$.

To show point (b) above is equivalent to showing that when restricted to W, we have $\mathsf{val}(\mathcal{E},\sigma) \sqsubseteq \mathsf{val}(\mathcal{A},\sigma)$ and $\mathcal{E}$ is a σ-respecting decomposition such that $\mathsf{val}(\mathcal{D},\sigma) \sqsubseteq \mathsf{val}(\mathcal{E},\sigma)$. The second part of the statement follows because if $\mathcal{E}$ is a σ-respecting decomposition whose decomposition labelling is larger than that of $\mathcal{D}$, and is also an attractor decomposition, from the closure under minimum of attractor decomposition (Lemma 6.2.1), we also get $\mathsf{val}(\mathcal{A},\sigma) \sqsubseteq \mathsf{val}(\mathcal{E},\sigma)$.

Showing $H_{\mathcal{E}}^{\eta} = H_{\mathcal{A}}^{\eta}$. The set H consisting of all vertices of priority d that have a path to a some cycle-winner, v such that $\mathfrak{L}_{\mathcal{D}}^{\sigma}(v)|_d = \eta$ is then identified. From Proposition 6.4.7, we know that for all $v \in H$, we have $\mathsf{val}(\mathcal{D}, \sigma)(v) \leqslant \mathsf{val}(\mathcal{A}, \sigma)(v) \leqslant (\eta, 0)$. Consider any path from v to u. Since by assumption, we know for all vertices w in $\mathcal{G}$ $\mathsf{val}(\mathcal{A}, \sigma)(w) \geqslant (\eta, 0)$, we know $\mathsf{val}(\mathcal{A}, \sigma)(v) = (\eta, 0)$ for all $v \in H$. Since $\mathcal{E}(v) = (\eta, 0)$, for all $v \in H$, we have $\mathsf{val}(\mathcal{A}, \sigma)(v) = \mathsf{val}(\mathcal{E}, \sigma)(v)$ for all $v \in H$.

We show that for any $u \notin H$, $\mathsf{val}(\mathcal{A}, \sigma)(v) > (\eta, 0)$. Consider any path from u, of priority $d = \text{Even-level}(\eta)$, consisting of only tight edges with respect to $\mathsf{val}(\mathcal{A}, \sigma)$ from some vertex valued at $(\eta, 0)$ by $\mathsf{val}(\mathcal{A}, \sigma)$. This leads to an even cycle, let w be a cycle-winner of such an even cycle. We show that $\mathfrak{L}_{\mathcal{D}}^{\sigma}(w)|_d \leqslant \eta$, which ensures that $u \in H$. Indeed for the path $u \to u_1 \to u_2 \to \cdots \to u_k = v$, where all edges are tight, for each i that we have $x_i|_d = x_{i+1}|_d$, where x_i is the first component of $\mathsf{val}(\mathcal{D}, \sigma)(u_i)$.

Showing $T_{\mathcal{E}}^{\eta} = T_{\mathcal{A}}^{\eta}$. T consists of vertices in the game that have a path in $(\mathcal{G}_\sigma)^{\leqslant d}$ to the set H. We know from Proposition 6.4.6 that $\mathsf{val}(\mathcal{D}, \rho)(u) \leqslant \mathsf{val}(\mathcal{A}, \sigma)(u) \leqslant \left(\eta^T, k\right)$ where k is the length of the shortest path in $\mathcal{G}^{\leqslant d}$ to some vertex in H.

We can also show $\mathsf{val}(\mathcal{A}, \sigma)(u) \geqslant \left(\eta^T, k\right)$. For this, we only need to follow tight edges with respect to $\mathsf{val}(\mathcal{A}, \sigma)$ from u. If these tight edges lead to some vertex such that $\mathsf{val}(\mathcal{A}, \sigma)(v) = (\eta, 0)$, then we know this path has length at least k and therefore $\mathsf{val}(\mathcal{A}, \sigma)(u) \geqslant \left(\eta^T, k\right)$. If the first skeleton node encountered by such a tight path is not η, and instead η', we know $\eta' > \eta$, since η is the root node of the tree, and hence $\mathsf{val}(\mathcal{A}, \sigma)(u) \geqslant \left(\eta', 0\right) > \left(\eta^T, k\right)$. This ensures that T is exactly the set of vertices valuated by $\mathsf{val}(\mathcal{A}, \sigma)$ to $\left(\eta^T, k\right)$ for some k. Since k corresponds to path lengths, it cannot be larger than $|V| - 1$.

Showing $\mathcal{E}_i = \mathcal{A}_i$. For each child η_i of η, we call $(W_i, \mathcal{E}_i)$ to be the output by Minimal-Improve$((\mathcal{G}_i)|_\sigma, \mathcal{D}, \mathcal{T}, \eta_i)$, We will show that if for all $i < j$ if

(a) $W_i = \left\{ w \mid \mathsf{val}(\mathcal{A}, \sigma)(w) \leqslant \left({\eta_i}^S, n-1\right) \right\}$ and (b) $\mathcal{E}_i|_{W_i} = \mathcal{A}_i|_{W_i}$,

then

(i) $W_j = \left\{ w \mid \mathsf{val}(\mathcal{A}, \sigma)(w) \leqslant \left({\eta_j}^S, n-1\right) \right\}$ and (ii) $\mathcal{E}_j|_{W_j} = \mathcal{A}_j|_{W_j}$.

For a base case, observe that $\mathsf{val}(\mathcal{A}, \sigma)(v) \geqslant (\eta_1, 0)$ for all vertices $v \in \mathcal{G}_1^{\leqslant d-1}$. From our induction hypothesis the above two conditions (i) and (ii) follow for $j = 1$.

For $j > 1$, notice that $\mathcal{G}_j^{\leqslant d-1}$ consists of vertices not in any W_i for $i < j$. Moreover, it consists of no vertex in $H_{\mathcal{A}}^{\eta}$ or $T_{\mathcal{A}}^{\eta}$. Hence, for $v_j \in \mathcal{G}_j^{\leqslant d-1}$, $\mathsf{val}(\mathcal{A}, \sigma)(v_j) \geqslant$

$(\eta_j, 0)$. And thus, $(W_j, \mathcal{E}_j) = \textsc{Minimal-Improve}((\mathcal{G}_i)|_\sigma, \mathcal{D}, \mathcal{T}, \eta_i)$ by induction hypothesis satisfies the above conditions.

Showing $S^\eta_\mathcal{E} = S^\eta_\mathcal{A}$. Observe that $S^\eta_\mathcal{E} = S$ which consists of vertices for which all paths to R visit a priority $d+1$ vertex. For each vertex $v \in S$, let k be the shortest length path to some u such that u in turn has a path that visits R without seeing a vertex of priority $d+1$. Indeed such a path must exist since all vertices in R themselves have priority at most d. We argue that $\mathsf{val}(\mathcal{D}, \rho)(v) \leqslant \mathsf{val}(\mathcal{A}, \sigma)(v) \leqslant \left(\eta^S, k\right)$.

For any u that has a path that visits R without seeing a vertex of priority $d+1$, we have $\mathsf{val}(\mathcal{D}, \rho)(u) = (x, i)$, where $x|_d \leqslant \eta$, and therefore u must be in either H, T or W_i for some i. Hence $\mathsf{val}(\mathcal{A}, \sigma)(u) \leqslant (x, i)$ for $x \in \mathcal{L}(\mathcal{T})$ such that $x|_d = \eta$.

Suppose the shortest path from v to u has length 1, then such a v must have priority $d+1$. Since $\mathsf{val}(\mathcal{A}, \sigma)(u) \leqslant (x, i)$ for $x \in \mathcal{L}(\mathcal{T})$ such that $x|_d = \eta$, this implies $\mathsf{val}(\mathcal{A}, \sigma)(v) \leqslant \left(\eta^S, 1\right)$.

The proof that a vertex which has a path of length k and not shorter path to some vertex u above satisfies $\mathsf{val}(\mathcal{A}, \sigma)(v) \leqslant \left(\eta^S, k\right)$ follows from routine induction.

Similar to the arguments for T, by following tight edges for some $v \in S$, we can also conclude that S is also exactly the set of vertices such that $\mathsf{val}(\mathcal{A}, \sigma) = \left(\eta^S, k\right)$ for some k. □

Chapter 7

An asymmetric attractor based algorithm

Progress-measure lifting algorithms for solving parity games have the better worst-case asymptotic time and space complexity [Jur00, JL17, FJdK+19, DS22] compared to the symmetric attractor-based algorithms [McN93, Zie98, LPSW22, JMT22]. On the other hand, attractor-based algorithms, and more specifically the McNaughton-Zielonka algorithm [McN93, Zie98], consistently outperform other algorithms in practice while having exponential running time in the worst-case [vD18, FJdK+19, BDM18]. A natural conjecture for the reason behind the success of several attractor-based algorithms, and especially McNaughton-Zielonka, could be attributed to repeated computation of attractors. Computing attractors in itself is a relatively fast operation and such computations performed during the algorithm can remove large sets of vertices resulting in smaller subgames to work with. However, attractor-based algorithms, even the quasi-polynomial ones, have a worst case running time that is approximately the square of the running time complexity of similar progress-measure-based algorithms.

In this chapter, we propose a new attractor-based algorithm with the aim of achieving practical efficiency through repeated computation of attractors, alongside possessing theoretical guarantees that align with state-of-the-art algorithms. But unlike other McNaughton-Zielonka-like algorithms, our algorithm is an "asymmetric" algorithm that builds only the attractor decomposition for one player. Our definition of an attractor decomposition is based on ordered trees, and for an appropriate universal tree $\mathcal{T}$, a parity game always has a $\mathcal{T}$-attractor decomposition. Our algorithm is parameterised by trees whose branching dictates the recursive calls made. Unsurprisingly, when instantiated with an appropriate universal tree $\mathcal{T}$, our

algorithm builds a $\mathcal{T}$-attractor decomposition. The running time complexity of our algorithm is linear in the size of such a tree underlying the game and polynomial in the size of the game. Our algorithm, when the underlying trees are complete trees, produces an exponential algorithm whose running time is comparable to the Jurdziński's progress measure algorithm (up to a polynomial factor), but computes the attractor decomposition output by (an enhanced version of) McNaughton-Zielonka. The same algorithm has worst-case running time that matches the running time (up to a polynomial factor) of state-of-the art algorithms like that of Jurdziński and Lazić [JL17] if the underlying tree is the Jurdziński-Lazić universal tree, or the runtime of our algorithm in Chapter 5 if the underlying tree is the Strahler Universal trees [DJT20]. In contrast, symmetric attractor based algorithms whose mutual recursive calls are dictated by similar recursive trees take time that is a square of such universal trees instead.

Our key ingredient to achieve this algorithm is using the notion of a decomposition of a game. This concept, as introduced in Chapter 6, is a relaxation of attractor decompositions. In our recursive algorithm, these decompositions are used to encode the progress made in previous recursive calls. Instead of restarting each recursive call from scratch, we use decompositions to expedite the process and serve as a succinct, yet robust encoding of the progress that was made in earlier recursive calls. This modification to the decomposition during each recursive call is done in a careful manner so as to satisfy monotonicity requirements which help us argue that our algorithm terminates faster than versions that do not use such decompositions.

The idea of reusing information from previous recursive calls for algorithms that solve parity games has been touched upon in works as early as the 1990s. In fact, Long, Browne, Clarke, Jha, and Marrero [LBC$^+$94] proposed the idea of utilising the information obtained from earlier recursive calls to aid subsequent recursive calls. They tackled the challenge of model checking modal μ-calculus formulas, a problem that is polynomial time equivalent to solving parity games. Their algorithm had a running time of $\mathcal{O}(n^{d/2})$, which was a quadratic improvement over the best algorithm obtained by solving parity games via McNaughton-Zielonka algorithm [McN93, Zie98, EL86]. Nonetheless, their approach suffered from exponential space complexity. Similar to our approach, they heavily rely on monotonicity arguments to demonstrate faster termination. However, our algorithm distinguishes itself by requiring only polynomial space, as well as performing repeated computation of attractors. Furthermore, when instantiated with quasi-polynomial universal trees, our algorithm achieves theoretical complexity comparable to the state of the art.

7.1 Finding attractor decompositions

Attractor decompositions are witnesses of winning for parity games and we begin by asking a natural question about them.

Question 7.1.1. *For a tree $\mathcal{T}$ and an (n, d)-parity game, can we compute a canonical $(\eta, \mathcal{T})$-attractor decomposition for $\mathcal{G}$?*

The above question without the word canonical is trivial, since we can create an $(\eta, \mathcal{T})$-attractor decomposition by just declaring most parts associated with such a decomposition to be empty to form a $\mathcal{T}$-attractor decomposition. Given a tree, we therefore want to find the "best" attractor decomposition, or one that identifies the largest possible winning set for this tree. We describe an algorithm that does that. The reader can find similarities between Algorithm 5 below and the McNaughton-Zielonka algorithm [McN93, Zie98]. More specifically, we remark without further proof that it closely resembles the Jurdziński-Morvan algorithm, where one tree is the complete tree and the other tree is the tree $\mathcal{T}$.

Algorithm 5 Computes an attractor decomposition of $\mathcal{G}$

Input: A game $\mathcal{G}$ (with maximum even priority d), a node η (of even level d) in the tree $\mathcal{T}$.
Output: An attractor decomposition of $\mathcal{G}$.

procedure AttractorDecomposition(η, $\mathcal{G}$)
  $S \leftarrow$all vertices in the Audrey attractor of all vertices of priority $d+1$
  $H^\eta \leftarrow$ all vertices in $\mathcal{G} \setminus S$ of priority equal to the level of η
  $T^\eta \leftarrow$ vertices in the strict Steven attractor to H^η in $\mathcal{G} \setminus S$
  $W \leftarrow T^\eta \cup H^\eta$
  for each child $\eta_1, \ldots, \eta_k$ of η in order **do**
    $\mathcal{A}_i \leftarrow$AttractorDecomposition(η_i, $\mathcal{G} \setminus (S \cup W)$)
    $W \leftarrow [\![\mathcal{A}_i^{\eta_i}]\!]$
    $W \leftarrow W \cup W_i$
  end for
  $A' \leftarrow$ Audrey attractor to $\mathcal{G} \setminus W$
  if $A' \cap W = \varnothing$ **then**
    $S^\eta \leftarrow$ vertices in the strict Steven attractor to W in $\mathcal{G}$
    $W \leftarrow W \cup S^\eta$
    return $\left(W, \left\langle H^\eta, T^\eta, (\mathcal{A}_1, \ldots, \mathcal{A}_k), S^\eta \right\rangle\right)$
  else
    return AttractorDecomposition(η, $\mathcal{G} \setminus A'$)
  end if
end procedure

For each node η of the tree, the set of vertices S from which Audrey has a

strategy to visit a priority vertex $d+1$ is first removed from $\mathcal{G}$. In the remaining game, the algorithm identifies some sets of the attractor decomposition the set H^η of vertices of the highest priority and the set: T^η of vertices that are in the (strict) Steven attractor to H^η. Excluding the above vertices, for each child η_i of η, we obtain recursively the $(\eta_i, \mathcal{T})$-attractor decomposition of the subgame induced a subset of vertices W_i.

The algorithm then checks if the set W consisting of H^η and T^η along with all the sets W_i forms a trap for Audrey. If W is indeed a trap for Audrey, then the set of vertices that are in the Steven attractor to W is computed and declared to be S^η. If instead Audrey can escape W, then the process is restarted after excluding these vertices.

The correctness of Algorithm 5 can be proved using a routine induction (on the number of nodes on the tree $\mathcal{T}$) that the algorithm returns a $\mathcal{T}$-decomposition $\mathcal{A} = \left\langle H^\eta, T^\eta, (\mathcal{A}_1, \ldots, \mathcal{A}_k), S^\eta \right\rangle$ of the largest dominion of $\mathcal{G}$ that has a $\mathcal{T}$-attractor decomposition.

The process terminates as each recursive call is made to a smaller subgame or, alternatively, with a smaller tree. Although we can show that this process terminates, we can also show that this procedure has an exponential worst case complexity. If the time taken for game with n vertices and a tree $\mathcal{T}$ which has height h and ℓ leaves is denoted by $R(n, h, \ell)$, then we can deduce that

$$R(n, h, \ell) \leqslant R(n-1, h, \ell) + R(n_1, h-1, \ell_1) + \cdots + R(n_k, h-1, \ell_k) + O(m)$$

where

- n_i denotes the number of vertices made in the call made in the for-loop with the root η_i, and
- ℓ_i denotes the number of leaves of the tree rooted at η_i in $\mathcal{T}$.

Moreover, we assume that if $\ell = 1$ or if $n = 0$, then $R(n, h, \ell) = 1$. Using the fact that the tree $\mathcal{T}$ has height $d/2$ along with a routine analysis of the above recurrence shows that the time taken can be bounded by $n^{d/2}|\mathcal{T}|$. In the next section, we propose a different, carefully engineered, attractor-based algorithm whose running time can be bounded instead by a function that is linear in the size of the tree and polynomial in n.

7.2 A faster attractor-based asymmetric algorithm

We propose our new procedure in Algorithm 6 where we build on the attractor decompositions obtained by earlier recursive calls instead. This is done by maintaining the decompositions obtained in successive recursive sub-calls and modifying them until we finally produce an attractor decomposition.

Given an $(n, d+1)$ parity game $\mathcal{G}$, a tree $\mathcal{T}$, and a node η of the tree that has an even level d, our algorithm computes the $(\eta, \mathcal{T})$-attractor decomposition of the largest dominion that has an $(\eta, \mathcal{T})$-attractor decomposition. The algorithm has an underlying $\mathcal{T}$-decomposition that it maintains globally, and each recursive call has access to this decomposition. We also assume that the first external call to this subroutine initialises the decomposition $\mathcal{D}$ where all vertices of priority $d+1$ are in $S^\eta_\mathcal{D}$, all vertices of priority d in $\mathcal{G}$ are in $H^\eta_\mathcal{D}$ and all the other vertices in $\mathcal{G}$ are in $T^\eta_\mathcal{D}$.

Algorithm 6 Computes the $\mathcal{T}$-attractor decomposition of the Steven dominion of $\mathcal{G}$

Input: A game $\mathcal{G}$ (with maximum even priority d), a node η (of even level d) in the tree $\mathcal{T}$. The first external call to this subroutine initialises the decomposition $\mathcal{D}$ where all vertices of priority $d+1$ are in $S^\eta_\mathcal{D}$, all vertices of priority d in $\mathcal{G}$ are in $H^\eta_\mathcal{D}$ and all the other vertices in $\mathcal{G}$ are in $T^\eta_\mathcal{D}$.
Output: An attractor decomposition of $\mathcal{G}$.

```
1:  procedure AAD(η, G)
2:      while D is not an attractor decomposition do
3:          U ← vertices in the Audrey attractor to S^η_D
4:          [D^η] ← [D^η] ⊖ U
5:          S^η_D ← S^η_D ∪ U
6:          T ← vertices in the Steven attractor to H^η_D in game [D^η]
7:          R ← T^η_D \ T
8:          Move_D(R)
9:          for each child η_1, ..., η_k of η ∈ T in order do
10:             AAD(η_i, [D^{η_i}])
11:         end for
12:         S ← vertices in the Steven attractor to [D^η]
13:         R' ← S^η_D \ S
14:         Move_D(R')
15:     end while
16: end procedure
```

The decomposition is iteratively modified by the algorithm as described below. In every iteration, in lines 4and 5, the algorithm removes the set of vertices U that is in Audrey's attractor to $S^\eta_\mathcal{D}$ in $\mathcal{G}$ and adds the vertices in U to $S^\eta_\mathcal{D}$. This step

ensures that the subgame $\left[\mathcal{D}^{\eta}\right]$ is an Audrey trap (the complement of an Audrey attractor is a trap for Audrey). The removal of U is denoted by $\left[\mathcal{D}^{\eta}\right] \leftarrow \left[\mathcal{D}^{\eta}\right] \ominus U$, which results in the removal of the set U from each of the parts of the decomposition contained in $\left[\mathcal{D}^{\eta}\right]$. The vertices in U are added instead to the side-set $S^{\eta}_{\mathcal{D}}$.

In line 6, the Steven attractor T to the vertices $H^{\eta}_{\mathcal{D}}$ with highest priority is computed. Observe that any vertex that is in $T^{\eta}_{\mathcal{D}}$ but not in T are vertices from which Steven does not have a strategy to visit the set of highest priority vertices $H^{\eta}_{\mathcal{D}}$. Hence, in line 7, vertices R that are currently in top-set $T^{\eta}_{\mathcal{D}}$ but not in the attractor T are identified, and in line 8 this set R is "moved". The modification done by Move is defined more rigorously later, but intuitively, after performing $\mathrm{Move}_{\mathcal{D}}(R)$ the decomposition $\mathcal{D}$ is changed so that the vertices in R are no longer in the part $T^{\eta}_{\mathcal{D}}$ and relocated to part S^{η} or to parts associated with the node γ which is greater than η. In line 10, recursively, the $(\eta_i, \mathcal{T})$ decompositions are computed for the complement of the game, one after another for each child η_i of the node η. In line 12, the Steven attractor is computed to the set of vertices that is currently in $\left[\mathcal{D}^{\eta}\right]$ and in line 13, the set R' of vertices in $S^{\eta}_{\mathcal{D}}$ but not in the Steven attractor is computed. These vertices are finally "moved" in the decomposition in line 14 and the vertices in R' are now associated either to the parts corresponding to the next sibling of η or the side-set of a parent of η if there is no next sibling.

7.2.1 An order between decomposition

Although we compute attractors, our algorithm can also be reformulated as a lifting algorithm on specific lattices designed for a parity game. We exploit this fact to show the algorithm's correctness and running time. The proofs of correctness of the algorithm and its termination indeed use monotonicity arguments based on a partial ordering of the set of all decompositions. Therefore, our key technical ingredient is to define this partial order (for a fixed tree $\mathcal{T}$ and a parity game $\mathcal{G}$) on the set of all $(\eta, \mathcal{T})$-Steven decompositions for all nodes η in $\mathcal{T}$. Henceforth, we assume decomposition to refer only to Steven decompositions by default.

In the previous chapter, we had given a valuation for a tuple consisting of decompositions $\mathcal{D}$ and an Audrey strategy σ for σ-respecting decompositions $\mathcal{D}$. But here we define a similar ordering on the set of all decompositions without reference to a strategy.

For a tree $\mathcal{T}$ in which all leaves have the same depth, recall the definition of a leafy tree of $\mathcal{T}$, denoted by $\mathcal{L}(\mathcal{T})$. We use the ordering of a leafy tree's nodes again to produce such a partial ordering on the set of all $\mathcal{T}$-decompositions. For each decomposition $\mathcal{D}$, we define the decomposition labelling of $\mathcal{D}$, denoted by $\delta_{\mathcal{D}}$,

as the following map from the vertices to the totally ordered set $\mathcal{L}(\mathcal{T}) \cup \{\top\}$:

$$\delta_{\mathcal{D}}(v) = \begin{cases} \gamma & \text{for } v \in H^{\gamma}, \\ \gamma^{T} & \text{if } v \in T^{\gamma}, \\ \gamma^{S} & \text{if } v \in S^{\gamma}, \\ \top & \text{if } v \notin H^{\gamma} \cup T^{\gamma} \cup S^{\gamma} \text{ for all } \gamma \in \mathcal{T}. \end{cases}$$

For a game, $\mathcal{G}$, and two $(\eta, \mathcal{T})$-decompositions $\mathcal{D}_1$ and $\mathcal{D}_2$, we say $\mathcal{D}_1 \sqsubseteq \mathcal{D}_2$ if $\delta_{\mathcal{D}_1} \sqsubseteq \delta_{\mathcal{D}_2}$, where $\sqsubseteq$ compares the decomposition labellings pointwise.

Observe that any map δ is the decomposition labelling of some decomposition $\mathcal{D}$ if for all v,

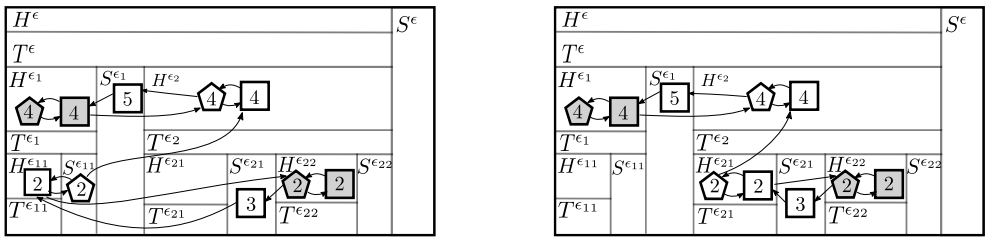
- $\pi(v) \leqslant \text{Even-level}(\delta(v))$ and
- if $\delta(v) = \eta \in \mathcal{T}$, then $\pi(v) = \text{Even-level}(\delta(v))$.

Furthermore, any $\mathcal{D}$ also has a decomposition labelling δ which also satisfies the above mentioned. The set of decompositions is in bijection with the set of decomposition labellings which satisfy the above property. We highlight that decomposition labellings are therefore a different representation of the same concept. Since the order of a decomposition is inherited from its associated labelling, which is a lattice, the set of decompositions forms a lattice.

Example 5. *In Fig. 7.1, we give two decompositions (similar to Fig. 6.3 in the previous chapter) to demonstrate the underlying order we have introduced among the decompositions. For the tree with three leaves, we see that the decomposition $\mathcal{D}$ in Fig. 7.1(a) is smaller than decomposition $\mathcal{E}$ in Fig. 7.1(b). This is because for the two vertices of priority 2 that are unshaded the decomposition $\delta_{\mathcal{D}}$ is pointwise at most as large as $\delta_{\mathcal{E}}$. Indeed the decomposition labelling $\delta_{\mathcal{D}}$ maps these vertices to nodes ϵ_{11} and ϵ_{11}^{S}, whereas the decomposition labelling $\delta_{\mathcal{E}}$ maps both the vertices to the node ϵ_{21}, which is strictly larger.*

Maps from vertices to leafy trees that are the decomposition labellings of attractor decompositions can be expressed using a combination of local properties (one-step progress) and global properties (attractors). Therefore, these maps resemble both progress measures and attractor decompositions. We call such maps attractor-decomposition labellings.

Proposition 7.2.1. *A map δ from the set of vertices of a parity game $\mathcal{G}$ to a leafy tree of $\mathcal{T}$ is the decomposition labelling corresponding to an $(\eta, \mathcal{T})$-Steven*



(a) Decomposition $\mathcal{D}$ (b) Decomposition $\mathcal{E}$

Figure 7.1: Two decomposition where $\mathcal{D} \sqsubseteq \mathcal{E}$

attractor decomposition $\mathcal{A}$ of $\mathcal{G}$ if and only if it satisfies the following conditions for all vertices v

- $\pi(v) \leqslant \text{Even-level}(\delta(v))$*;*
- *if $\delta(v) = \eta$ and $\eta \in \mathcal{T}$, then $\pi(v) = \text{Even-level}(\delta(v))$ and there is a one-step strategy for Steven to visit a vertex u such that $\delta(u) < \eta^S$;*
- *if $\delta(v) = x$ and $x = \eta^T$ or $x = \eta^S$, then there is a reachability strategy from u to vertices in $\{u \mid \delta(u) < x\}$ that avoids any vertex from $\{u \mid \delta(u) > x\}$.*

This ordering on attractor decompositions is robust as the set of all attractor decompositions is closed under taking the minimum. We show this in Proposition 7.2.2 using Proposition 7.2.1, thus showing that attractor decompositions form a semi-lattice under the order $\sqsubseteq$. More importantly, for a tree $\mathcal{T}$, there is a unique minimum $\mathcal{T}$-attractor decomposition of the game $\mathcal{G}$.

Proposition 7.2.2. *For an (n, d)-small parity game $\mathcal{G}$, a tree with even level d, and two $\mathcal{T}$-attractor decompositions $\mathcal{A}_1$ and $\mathcal{A}_2$ of dominions D_1 and D_2, respectively, of $\mathcal{G}$, the minimum decomposition $\mathcal{A}$ of both is an attractor decomposition $\mathcal{A}$ of the dominion $D_1 \cup D_2$ of $\mathcal{G}$.*

Proof. Consider $\mathcal{D}$ to be the decomposition obtained from the point-wise minimum of the decomposition labelling δ_1 and δ_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. This ensures that $\mathcal{D}$ thus defined is a minimal decomposition. All that remains is to show that it is an attractor decomposition, which we do by inductively (inducting on the height of the tree). We only state that for trees with height 0, the proof is routine.

We show inductively that $\mathcal{D}$ is an $(\eta, \mathcal{T})$-attractor decomposition.

Induction hypothesis. If for all v, since $\delta(v) = \min\{\delta_1(v), \delta_2(v)\} \geqslant \eta$, then $\mathcal{D}$ so defined is an $(\eta, \mathcal{T})$-decomposition.

We only need to show that δ satisfies the conditions mentioned in Proposition 7.2.1. We list the conditions as in the proposition and show that they are satisfied.

- $\pi(v) \leqslant \text{Even-level}(\delta(v))$ and more specifically we also have if $\delta(v) = \eta$ for $\eta \in \mathcal{T}$, then $\pi(v) \leqslant \text{Even-level}(\delta(v))$.
- Assume without loss of generality that $\delta_1(v) = \delta(v)$.
  - v belongs to Steven, then we know that there is *some* neighbour u such that $\delta(u) \leqslant \delta_1(u) < \eta^S$.
  - v belongs to Audrey, the same argument works for *all* of its neighbours.
- If $\delta(v) = x$ and $x = \eta^S$ or $x = \eta^T$, then, without loss of generality, there is a reachability strategy for Steven from u to vertices in $\{u \mid \delta_1(u) < x\}$ that avoids any vertex from $\{u \mid \delta_1(u) > x\}$. Observe that in the set $\{u \mid \delta_1(u) < x\} \subseteq \{u \mid \delta(u) < x\}$ and $\{u \mid \delta_1(u) > x\} \supseteq \{u \mid \delta(u) > x\}$. So, the same reachability strategy as that of Steven would work for the decomposition labelling δ. □

7.2.2 A discussion on Move

We describe Algorithm 8 using $\text{Move}_{\mathcal{D}}(R)$, which is defined only when the set R is in the same part, either S^γ or T^γ, for some node γ of the decomposition $\mathcal{D}$. The subroutine increases the value of the decomposition by modifying it so that only the vertices in R are rearranged. But the increase should ensure that it is still smaller than the smallest attractor decomposition. We provide a way to implement this operator below in the proof of Proposition 7.2.3. Here, we instead take the view of how each decomposition corresponds to decomposition labelling and then modify it so that it increases minimally in its valuations at R and nowhere else.

Proposition 7.2.3. *The modification by the subroutine* $\text{Move}_{\mathcal{D}}(R)$ *of a decomposition* $\mathcal{D}$ *can be implemented in nearly linear time.*

Proof. We assume that the decomposition is represented by its corresponding decomposition labelling. We first define a useful operator $\oplus$ for a subset of vertices R, and a Steven decomposition $\mathcal{D}$ and a node η at even level p. We use as shorthand $[\![\mathcal{D}^\eta]\!] \oplus R$ to denote the following sequence of modifications to $\mathcal{D}$

- $H^\eta_{\mathcal{D}} \leftarrow H^\eta_{\mathcal{D}} \cup \left(R \cap \pi^{-1}(p)\right)$,
- $T^\eta_{\mathcal{D}} \leftarrow H^\eta_{\mathcal{D}} \cup \left(R \cap \pi^{-1}(< p)\right)$, and

- $S^{\eta}_{\mathcal{D}} \leftarrow H^{\eta}_{\mathcal{D}} \cup \left(R \cap \pi^{-1}(p+1)\right)$.

We define smallMove$_{\mathcal{D}}(R)$, where we have $R \subseteq T^{\eta}_{\mathcal{D}}$, as the following operator

- $T^{\eta}_{\mathcal{D}} \leftarrow T^{\eta}_{\mathcal{D}} \setminus R$.
- If η has a child and let η_1 be the first child, whose even level is p then $[\![\mathcal{D}^{\eta_1}]\!] \oplus R$.
- If η is a leaf, then we re-assign the vertices to the side-set $S^{\eta}_{\mathcal{D}} \leftarrow S^{\eta}_{\mathcal{D}} \cup R$.

For $R \subseteq S^{\eta}_{\mathcal{D}}$, we instead modify the decomposition as follows:

- $S^{\eta}_{\mathcal{D}} \leftarrow S^{\eta}_{\mathcal{D}} \setminus R$.
- If η is the last child of its parent γ, then $S^{\gamma}_{\mathcal{D}} \leftarrow S^{\gamma}_{\mathcal{D}} \cup R$.
- If not, then let γ be the sibling of η and modify $\mathcal{D}$ by $[\![\mathcal{D}^{\gamma}]\!] \oplus R$.

Since the relabelling of vertices in R requires the algorithm to only compute the next sibling or parent, we assume such operations take at most linear time. □

We state here that Move$_{\mathcal{D}}(R)$ modifies the decomposition and satisfies the following properties:

1. it is larger than $\mathcal{D}$;
2. all vertices not in R belong to the same parts as in the decomposition $\mathcal{D}$;
3. all vertices in R are not in the same parts as in decomposition $\mathcal{D}$;
4. it is no larger than the smallest attractor decomposition larger than $\mathcal{D}$.

Note that many different decompositions can satisfy the above properties. We emphasise that *any* decomposition that satisfies the above four properties is sufficient to prove correctness, but restrict ourselves to the one explicitly constructed in Proposition 7.2.3.

We also take this opportunity to remark that one can view our algorithm as a progress measure algorithm, however, restricted to very specific kind of ordered trees. In fact, decomposition labellings capture exactly such measures. The striking difference is that progress measure algorithms are generic frameworks in which one can use any policy to decide the order in which to lift vertices, whereas ours dictates a rigid order on the set of vertices (rather than a specific vertex) that need to be lifted. On the other hand, we regain some flexibility, since we are allowed to moderate by how much a vertex is lifted. This is captured by our description of a Move subroutine that is only required to satisfy items 1-4.

7.2.3 Correctness and running time

We state our main theorem, to prove which we require Lemmas 7.2.4 and 7.2.7.

Theorem E. *For a parity game $\mathcal{G}$ and a tree $\mathcal{T}$, Algorithm 6 (on page 106) takes time at most linear in the number of nodes in $\mathcal{T}$ and polynomial in the size of the game $\mathcal{G}$ to produce the largest Steven dominion that has a Steven $\mathcal{T}$-attractor decomposition.*

The correctness of our algorithm follows from Lemma 7.2.4, which, in turn, uses Lemma 7.2.5. The latter states that each iteration of the while loop increases the underlying decomposition. This lemma is also key to concluding our desired runtime.

Later in Lemma 7.2.7, we argue that at most $\mathcal{O}(md)$ time pases before there is a change in the underlying decomposition. Since there are only $n|\mathcal{T}|$ many decompositions, this ensures that the algorithm terminates in time $\mathcal{O}(nmd|\mathcal{T}|)$. Therefore, we also get our desired running time, which is at most $\mathcal{O}(nmd|\mathcal{T}|)$. The rest of the section is dedicated to the proof of Theorem E.

Lemma 7.2.4. *For an (n,d)-small parity game $\mathcal{G}$, a tree $\mathcal{T}$ and a node η whose level is at least as large as d, procedure* $\mathrm{AAD}(\eta,\mathcal{G})$ *in Algorithm 6 returns the smallest $(\eta,\mathcal{T})$-attractor decomposition of $\mathcal{G}$.*

Proof. To prove correctness of the algorithm and the above theorem, we prove a stronger technical lemma from which we can obtain the theorem as a corollary. Observe that we assume that the algorithm initialises the decomposition to the smallest decomposition. For a decomposition $\mathcal{D}$, we show in Lemma 7.2.5 that one iteration of the outermost loop modifies the decomposition in a way that this decomposition increases, whilst remaining smaller than the smallest attractor decomposition larger than $\mathcal{D}$. Since the while loop terminates only when the decomposition $\mathcal{D}$ is an attractor decomposition, we can conclude from the above that the decomposition obtained at the end is the smallest attractor decomposition larger than $\mathcal{D}$. Since the algorithm starts with the smallest $\mathcal{D}$, it terminates with the smallest $(\eta,\mathcal{T})$-attractor decomposition $\mathcal{A}$ of $\mathcal{G}$. □

Lemma 7.2.5. *For an (n,d)-small parity game $\mathcal{G}$, a tree $\mathcal{T}$ and a node η whose level is at least as large as d, if $\mathcal{D}$ is the decomposition at the beginning of each iteration of the procedure* $\mathrm{AAD}(\eta,\mathcal{G})$ *in Algorithm 6, then at the end of the iteration, the decomposition maintained is always:*

- *strictly larger than $\mathcal{D}$, and*

- *smaller than $\mathcal{A}$, which is the smallest attractor decomposition larger than $\mathcal{D}$.*

Proof. We proceed inductively on the sum of the number of nodes in the subtree of $\mathcal{T}$ rooted at η and the number of vertices in $\mathcal{G}$. If the number of vertices is 0, then trivially $\mathcal{D}$ forms an attractor decomposition of $\mathcal{G}$.

Since we use monotonicity properties to argue correctness in our proofs, it is more natural to refer to the corresponding decomposition-labellings of the computed decompositions. Let $\mathcal{A}$ denote the smallest decomposition larger than starting decomposition $\mathcal{D}$, and let α denote the decomposition labelling of $\mathcal{A}$. The overall structure of the proof is to argue that each modification done to the algorithm satisfies the conditions given in the lemma.

Lines 4 and 5. The algorithm first computes the Audrey attractor to vertices in S^η in $\mathcal{G}$. Let δ be the decomposition labelling of such a decomposition. If there is no intersection between $\left[\mathcal{D}^\eta\right]$ and U, then note that $\left[\mathcal{D}^\eta\right]$ forms a trap. If not, and there is an intersection, we show that for all vertices $u \in U \setminus S^\eta$ in the intersection, $\alpha(u) \geqslant \eta^S$ in Proposition 7.2.6.

Modification done by Line 8. The attractor T to the set of highest priority vertices in $\left[\mathcal{D}^\eta\right]$ is computed. The set of vertices $R = T^\eta_{\mathcal{D}} \setminus T$ is first identified, and the operation $\mathrm{Move}_R(\mathcal{D})$ is performed. If $T^\eta_{\mathcal{D}} = T$, then no change is made to the decomposition $\mathcal{D}$ as $R = \varnothing$. If not, recall that as defined earlier, Move modifies $\mathcal{D}$ to a decomposition $\mathcal{E}$ that is larger than $\mathcal{D}$, all the vertices not in R are left undisturbed, and all the vertices in R are reallocated to a different partition such that the decomposition obtained is larger. Since for all $v \in R$, we must have $\alpha(v) > \eta^T$ due to no Steven strategies to reach $T^\eta_{\mathcal{D}} = \{w \mid \delta(w) < \eta^S\}$ without visiting any vertices in $\{w \mid \delta(w) > \eta^S\}$.

The For loop. For each child η_i of η, we know from inductive hypothesis that $\mathrm{ADD}(\eta_i, \mathcal{T}, \left[\mathcal{D}^{\eta_i}\right])$ ensures that the decomposition after the procedure is an $(\eta_i, \mathcal{T})$-attractor decomposition when restricted to the set of vertices in $\left[\mathcal{D}^{\eta_i}\right]$. Additionally, we know that the decomposition does not increase more than the attractor decomposition for the subgame $\left[\mathcal{D}^\eta\right]$, and therefore for the whole game.

Modification done by Line 14. Finally, the attractor S to the set $\left[\mathcal{D}^\eta\right]$ is computed. The set of vertices $R' = S^\eta_{\mathcal{D}} \setminus S$ is identified. Since there is no Steven attractor strategy from the set of vertices currently assigned by δ to η^S, for all

$v \in R'$, we must have $\alpha(v) > \eta^S$. Note that this is exactly the operation $\text{Move}_{R'}(\mathcal{D})$ performs. □

Proposition 7.2.6. *For an $(\eta, \mathcal{T})$-decomposition $\mathcal{D}$ of a parity game $\mathcal{G}$, if Audrey has a strategy to visit $S^\eta_\mathcal{D}$ from all vertices $U \subseteq \left[\mathcal{D}^\eta\right]$, then for any $(\eta, \mathcal{T})$-attractor decomposition $\mathcal{A}$ that is larger than $\mathcal{D}$, then U does not intersect with $\left[\mathcal{A}^\eta\right]$*

Proof. Since $\mathcal{A}$ is an $(\eta, \mathcal{T})$-attractor decomposition of $\mathcal{G}$ by definition $\left[\mathcal{A}^\eta\right]$ forms a trap in $\mathcal{G}$.

We show our proposition by induction on the smallest number of steps in which Audrey can force the visit to S^η. It is true that all vertices u in $\mathcal{G}$ with a zero-step reachability strategy to S^η have $u \notin \left[\mathcal{A}^\eta\right]$. It is trivial that $u \notin \left[\mathcal{D}^\eta\right]$, and since $\mathcal{A}$ is larger than $\mathcal{D}$, it follows.

We show that if all vertices u where Audrey has an at-most i-step reachability strategy to visit S^η are such that $u \notin \left[\mathcal{A}^\eta\right]$, then so do all vertices v where Audrey has an $i+1$-step reachability strategy have $u \notin \left[\mathcal{A}^\eta\right]$.

Consider vertices with an $i+1$-step Audrey strategy to visit S^η. For all such $u \in U$ with an $i+1$-step strategy, Audrey can ensure a visit to a vertex v with an at most i-step Audrey strategy to S^η. For such v, we know $v \notin \left[\mathcal{A}^\eta\right]$ by induction.

If Audrey can visit v from u, since $v \notin \left[\mathcal{A}^\eta\right]$, the subgame formed by $\left[\mathcal{A}^\eta\right]$ would not be a trap, and hence $u \notin \left[\mathcal{A}^\eta\right]$. □

Lemma 7.2.7. *For an (n, d)-small parity game $\mathcal{G}$, a tree $\mathcal{T}$ and a node η in $\mathcal{T}$ whose level is at least as large as d, the procedure $\text{AAD}(\eta, \mathcal{G})$ in Algorithm 6 takes at most $\mathcal{O}(md)$ time before modifying the decomposition.*

Proof. If the current decomposition during a call of the procedure with node η is already an $(\eta, \mathcal{T})$-attractor decomposition, then this subcall takes at most time $\mathcal{O}(md)$ time to check if our decomposition is an attractor decomposition. The crux of our argument boils down to showing that we spend at most $\mathcal{O}(md)$ time before modifying the decomposition $\mathcal{D}$ such that it increases.

We call a recursive subcall $\text{AAD}(\eta, \mathcal{G})$ *trivial* if it is made to an empty subset of vertices. We assume that there are no trivial subcalls made by the algorithm. This can be assumed if the decomposition is maintained as a labelling by the algorithm. In fact, we store for each vertex of the game an element of the leafy tree of $\mathcal{T}$. The algorithm, instead of iterating through all the children of the tree, only makes a recursive call at a node η when there is some vertex all which has a labelling corresponding to a descendant of η.

We address two things; in a non-trivial subcall, the amount of work done outside this recursive call is at most $\mathcal{O}(md)$, and secondly, at most $\mathcal{O}(md)$ time passes before some modification is made to the underlying decomposition.

The fact that the algorithm outside of the recursive calls takes time $\mathcal{O}(md)$ is because the algorithm either computes an attractor to a set, performs basic set operations, or calls the Move subroutine. Computing attractors itself takes time at most $\mathcal{O}(m)$, but since we store decompositions as labellings, accessing a subset of vertices takes time $\mathcal{O}(md)$, and so does performing the Move subroutine, thus contributing to the runtime claimed above.

Now, we show that not too much time is spent between two modifications to the decomposition. If, at the beginning of the procedure, the decomposition $\mathcal{D}$ is not already an attractor decomposition, then we know that one of the following is true.

- if $\delta(v) = \gamma$ and there is a no one-step strategy for Steven to visit a vertex u such that $\delta(u) < \gamma^S$ or
- if $\delta(v) = x$ where $x = \gamma^T$ or γ^S, then there is an Audrey reachability strategy from u to $\{v \mid \delta(v) > x\}$.

Let v be the smallest vertex with such a δ. We argue that we make at most nd many (non-trivial) recursive subcalls in a row, taking a total time of $\mathcal{O}(md)$ before modifying the decomposition $\mathcal{D}$ that results in changing the value of the decomposition labelling of v. This is true, since for some γ, a descendant of η, $\delta(v) = \gamma \in \mathcal{T}$ or $\delta(v) = \gamma^T$ or $\delta(v) = \gamma^S$. No modification of the decomposition is made until the recursive call is made at node γ due to the minimality of v. This ensures that all recursive calls made at node γ' that are not to an ancestor of γ terminate 'quickly', due to all the vertices already forming a $(\gamma', \mathcal{T})$-attractor decomposition. The total time taken is bounded by $\mathcal{O}(md)$, as each non-trivial recursive subcall made corresponds to a disjoint set of vertices that already form an attractor decomposition. The sum of the time taken to check that the decomposition, when restricted to these vertices, actually forms an attractor decomposition is bounded above by $\mathcal{O}(md)$. □

7.3 Discussion

Our algorithm runs in time that is linear in the size of the input tree. Hence, this algorithm performs better asymptotically when the input parity games have attractor decompositions whose tree has size that is comparable to the number of vertices in the parity game. Two such scenarios can be imagined where the

tree of attractor decomposition is significantly smaller than the Jurdziński-Lazić universal trees required to solve the game. If the parity game has a small (or large) Strahler number, we have demonstrated in Chapter 5 that the trees required to solve them are polynomial in the size of the game. Another promising case is where the attractor decomposition tree is small because most vertices are in the attractor sets corresponding to the attractor decomposition. In such cases, although the number of vertices in the underlying game is large, the trees required to solve them might be much smaller.

If we are to solve arbitrarily large parity games, we suffer from several pitfalls of progress measure algorithms. If the game is winning for Audrey from everywhere and we run our algorithm, which computes the attractor decomposition for Steven, then our algorithm exhaustively rules out all Steven decompositions before concluding that the game is losing for Steven. Such instances, which exhibit the worst-case complexity are caused by the lack of a symmetric treatment of the players in the algorithm. In the next chapter, we produce an algorithm that is recursive and attractor-based and also ensures a symmetric treatment of the players. We rely heavily on the techniques developed in this chapter to develop our new algorithm. Our symmetric algorithms follow the same tree of recursive calls as several preexisting symmetric attractor based algorithms, but with a square root of their worst-case complexity. Consequently, our algorithm can be seen as an enhancement to these algorithms, which ultimately leads to an improvement in their runtimes.

Chapter 8

Symmetric attractor-based algorithm

Among the first algorithms to solve parity games were symmetric attractor-based algorithms. Based on the algorithm of McNaughton [McN93] to solve Muller games, Zielonka [Zie98] proposed an algorithm to solve parity games in his influential work. The runtime complexity of the McNaughton-Zielonka algorithm was $O(n^d)$, for an (n, d)-small parity game. Although there are currently several algorithms that boast a better worst-case runtime complexity, this algorithm remains among the fastest in practice [FL09, LPSW22, FJdK$^+$19].

We propose a new algorithm that is a symmetric attractor-based algorithm for solving parity games. Ours can be seen as a technique to enhance other symmetric attractor-based algorithms, to bring their theoretical complexities close to the state of the art. We focus on illustrating this technique using the example of the classic McNaughton-Zielonka algorithm [Zie98] and we also argue that it is applicable to other symmetric attractor-based algorithms that were inspired by McNaughton-Zielonka [BDM18, Par19, LPSW22, JMT22, LBD20]. McNaughton-Zielonka and its variants have exhibited excellent performance in practice, significantly beating other classes of algorithms on standard benchmarks [Kei15, vD18, BDM18, BDM$^+$21]. On the other hand, their worst-case asymptotic running time is typically worse than that of asymmetric algorithms [Jur00, CJK$^+$22, JL17, DJT20]. More specifically, while the running time of state-of-the-art asymmetric algorithms is dominated by the size of an underlying universal tree [JL17, DJT20], it is the square of the size of a universal tree for symmetric algorithms [JMT22, LPSW22]. We reduce the worst-case running time of symmetric attractor-based algorithms to match the linear dependence on the size of a universal tree enjoyed by asymmetric algorithms.

Our algorithm is based on making better use of the structural information obtained from earlier recursive calls. This significantly reduces the worst-case overall size of the tree of recursive calls of the algorithm. While existing symmetric attractor-based algorithms are typically computing just the winning sets or positional winning strategies, we propose to enhance them to explicitly record decompositions, which are finely structured witnesses of winning strategies introduced in the previous chapters. Moreover, we show how decompositions for *both* players obtained from recursive calls on subgames can be meaningfully used to reduce the sizes of subgames on which further recursive calls are made, even if their key properties are damaged by the removal of some vertices from subgames on which they were computed. In contrast, other symmetric attractor-based algorithms are wasteful by routinely discarding witnesses for one of the players that are computed in recursive calls; in the worst case, this results in repeatedly solving large subgames from scratch.

Our technique is robust and applies to both the classic exponential-time McNaughton-Zielonka algorithm [Zie98] and its more recent quasi-polynomial variants [Par19, JMT22, LPSW22]. We are also confident that it is applicable to other symmetric attractor-based algorithms, such as priority promotion [BDM18], which are variants of the McNaughton-Zielonka algorithm. Such algorithms can be interpreted as the enhancements by ad hoc heuristics to be robust to the wasteful behaviour described above. Ours is a more principled approach, in which decompositions of subgames computed in previous recursive calls are never discarded and are instead used in a systematic manner to speed up and reduce the number of further recursive calls.

8.1 An exponential-time symmetric algorithm

Although McNaughton-Zielonka algorithm outperforms several other algorithms in practice, there are several families of games on which it takes exponential time. Some examples are those found in the paper of Friedmann [Fri11], Gazda and Willemse [GW13], van Dijk [vD18], Benerecetti et al [BDM20]. We add to this list a family of games on which McNaughton-Zielonka makes exponentially many recursive calls. We focus our attention in this section to this family and use it as a running example to highlight the exponential complexity of McNaughton-Zielonka, to motivate the idea behind our technique, and to show how our technique leads to significant improvement.

We recall the classic recursive algorithm of McNaughton and Zielonka [McN93,

Zie98]. However, this is not similar to a description of the McNaughton-Zielonka algorithm that one would find in the wild. It is enhanced to produce attractor decompositions for both of the players and to return the winning sets for Steven and Audrey. The work of Jurdziński and Morvan [JM20, JMT22] also contains such an enhanced version of the McNaughton-Zielonka algorithm that produces attractor decompositions. We reproduce it with modifications in Algorithm 7 to align with the current definition of attractor decompositions.

The algorithm, on an (n, h)-small parity game $\mathcal{G}$, uses two complete n-ary trees of even level and odd level h and $h+1$, which dictates its recursive calls. The algorithm uses two mutually recursive calls McNZ-Even and McNZ-Odd, which takes as input a game $\mathcal{G}$, the highest priority h, and two nodes ϵ and ω from the two complete trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$. The node ϵ belongs to Steven's n-ary tree $\mathcal{T}^{\text{Even}}$ and the node ω to Audrey's n-ary tree $\mathcal{T}^{\text{Odd}}$. The respective levels of these nodes ϵ and ω in the tree are h and $h+1$.

The trees $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$ are used to construct the $(\epsilon, \mathcal{T}^{\text{Even}})$-attractor decomposition for Steven and $(\omega, \mathcal{T}^{\text{Odd}})$-attractor decomposition for Audrey. We assume that the algorithm stores the attractor decompositions $\mathcal{D}_{\text{Even}}$ and $\mathcal{D}_{\text{Odd}}$ computed in previous recursive calls. As in the previous chapter, these attractor decompositions are stored as a partition of the set of vertices, accessed using the nodes of the complete trees. These disjoint parts are referred to as $H^{\epsilon}_{\text{Even}}$, $T^{\epsilon}_{\text{Even}}$ and $S^{\epsilon}_{\text{Even}}$ for each node ϵ in the n-ary Steven's tree $\mathcal{T}^{\text{Even}}$ and H^{ω}_{Odd}, T^{ω}_{Odd}, and S^{ω}_{Odd}, for ω in Audrey's tree $\mathcal{T}^{\text{Odd}}$. We exclusively use ϵ and its variants for Steven's trees and ω and its variants for Audrey's trees to avoid confusion. In this section, the tree is also assumed to be $\mathbb{N}$-labelled, and we write $\langle x_1, \ldots, x_\ell \rangle \odot y$ for the tuple $\langle x_1, \ldots, x_\ell, y \rangle$ obtained by concatenating $\langle x_1, \ldots, x_\ell \rangle$ with $\langle y \rangle$. Recall that if a tree is assumed to have a *natural labelling* (labelled using $\mathbb{N} \setminus \{0\}$), we then represent the i^{th} child of a node η by $\eta \odot i$.

We start by describing the McNaughton-Zielonka algorithm modified to produce an $(\epsilon, \mathcal{T}^{\text{Even}})$-attractor decomposition for Steven and an $(\omega, \mathcal{T}^{\text{Odd}})$-attractor decomposition for Audrey. The algorithm runs iteratively until an attractor decomposition is found. In iteration i, the Steven attractor T_i to the set of vertices of highest even priority H_i is found, and removed from the current game $\mathcal{G}_i$. Then, Audrey's version of the procedure is called on the subgame $\mathcal{G}'_i$, obtained from $\mathcal{G}_i$ after the removal of T_i. Once the corresponding winning set U'_i of Audrey in $\mathcal{G}'_i$ is computed, the $(\omega \odot i, \mathcal{T}^{\text{Odd}})$-decomposition for Audrey is found except for the side-attractor set S^{ω}_{Odd}. To identify this set, Audrey's attractor S'_i to this Audrey-winning dominion U'_i is computed and the statement $S^{\omega \odot i}_{\text{Odd}} \leftarrow A'_i \setminus U'_i$ in line 13

declares the side attractor set for Audrey. The next subgame $\mathcal{G}_{i+1}$ is obtained by removing S_i' from $\mathcal{G}_i$.

The above process is repeated until the Audrey's attractor S_i to her winning set U_i does not contain any vertices other than U_i. If this is the case, we know that Steven can win from all vertices of $\mathcal{G}_{i+1}$ computed, and the statements $H^{\epsilon}_{\text{Even}} \leftarrow H_i$ and $T^{\epsilon}_{\text{Even}} \leftarrow T_i \setminus H_i$ fix the highest priority set and top attractor set, respectively, in lines 16 and 17. In the Steven recursive call, we only identify the partitions $H^{\epsilon}_{\text{Even}}$, $T^{\epsilon}_{\text{Even}}$ for Steven along with the parts $S^{\omega \odot i}_{\text{Odd}}$ for Audrey. The other parts of the partition are filled out by recursive calls of the procedure. We only describe the

Algorithm 7 McNaughton and Zielonka enhanced to produce attractor decompositions of both players

Input: A parity games $\mathcal{G}$ with highest priority h and nodes ϵ in the even tree $\mathcal{T}^{\text{Even}}$ whose even level is h and ω in the odd tree $\mathcal{T}^{\text{Odd}}$ whose odd level is $h+1$.
Output: The set of winning vertices for Steven.

1: **procedure** McNZ-Even($\mathcal{G}, h, \omega, \epsilon$)
2: **if** $\mathcal{G} = \varnothing$ **then**
3: **return** $\varnothing$
4: **end if**
5: $\mathcal{G}_1 \leftarrow \mathcal{G}$; $i = 0$
6: **repeat**
7: $i \leftarrow i + 1$
8: $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$
9: $T_i \leftarrow$ Steven attractor to H_i in $\mathcal{G}_i$
10: $\mathcal{G}_i' \leftarrow (\mathcal{G}_i \setminus T_i)$
11: $U_i' \leftarrow$McNZ-Odd($\mathcal{G}_i', h-1, \epsilon, \omega \odot i$)
12: $S_i' \leftarrow$ Audrey attractor to U_i' in $\mathcal{G}_i$
13: $S^{\omega \odot i}_{\text{Odd}} \leftarrow S_i' \setminus U_i'$
14: $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus S_i'$
15: **until** $S^{\omega \odot i}_{\text{Odd}} = \varnothing$
16: $H^{\epsilon}_{\text{Even}} \leftarrow H_i$
17: $T^{\epsilon}_{\text{Even}} \leftarrow T_i \setminus H_i$
18: **return** $V(\mathcal{G}_{i+1})$
19: **end procedure**

recursive calls at the even levels as the other recursive for odd levels can be described similarly. The correctness of the above algorithm is well known, and we refer an interested reader to several preexisting works [Zie98, JPZ08, JMT22] proving it.

In the upcoming discourse, we produce an example family of games for which McNaughton-Zielonka's algorithm exhibits exponential time complexity (Section 8.1.1). Subsequently, we introduce our enhancement in Algorithm 8, which has a worst-case running time is the square root of the worst-case running time of

McNaughton-Zielonka algorithm (Section 8.1.2). By proving polynomial-time termination (Section 8.1.3) of our algorithms for these specific game families, we claim a significant advantage over the McNaughton-Zielonka approach, which, when applied to these identical game sets, exhibits exponential time complexity.

8.1.1 An exponential family of games for McNaughton-Zielonka

Consider the following family of games $\mathcal{H}_k$ for each k. The game $\mathcal{H}_k$ contains of $5k$ vertices, with the highest priority as $k+2$. For all $i \leqslant k$, the vertex set contains 5 vertices and they are $\{u_i, v_i, w_i, x_i, y_i\}$. We call this set L_i and refer to it as the i^{th} layer of the game. The priority of w_i is $i+2$ and all the other nodes in L_i have priority $i+1$. For even values of i, Audrey owns v_i, y_i and Steven owns u_i, x_i. For odd values of i, this is swapped. The vertex w_i is owned by Steven for all i. The edges within a layer L_i are: $\{(u_i, v_i), (v_i, u_i), (v_i, x_i), (x_i, w_i), (w_i, v_i), (x_i, y_i), (y_i, x_i)\}$. Between layers,

- for each $i \leqslant k-2$, there is an edge (u_i, y_{i+2});
- for each $1 < i \leqslant k$, there are edges (v_i, v_{i-1}) and (y_i, y_{i-1}).

An example of the game $\mathcal{H}_4$ is shown in Figure 8.1, where the square vertices are owned by Steven and the pentagon ones by Audrey. The odd layers in the game are winning for Steven, whereas the even layers are winning for Audrey. A strategy for each player is to move to the vertex to the left. More formally, for each player, including all their edges (v_j, u_j) and (y_j, x_j) (along with all the edges of the opponent player) turns out to be a winning strategy in their respective dominions.

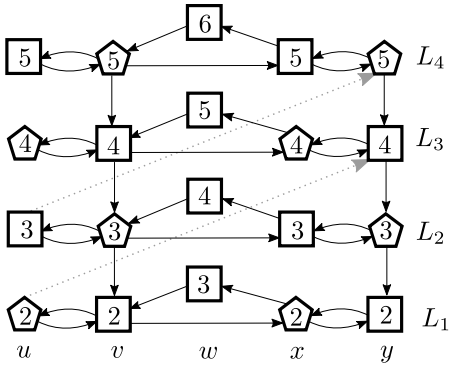


Figure 8.1: The game $\mathcal{H}_4$

Lemma 8.1.1. *For the family of games $\mathcal{H}_n$ for $n \geqslant 1$ Algorithm 7 makes $\Theta(2^n)$ recursive calls.*

We first highlight the idea behind the exponential complexity of McNaughton-Zielonka. For an odd value k, the procedure McNZ-Even on $\mathcal{H}_k$ makes two McNZ-Odd calls and these are made on the subgames $\mathcal{H}_{k-1} \cup \{u_k, v_k, y_k\}$ and $\mathcal{H}_{k-1} \cup \{x_k, y_k\}$ in succession. Notice that these games have a large intersection, which includes the subgame $\mathcal{H}_{k-1}$ and the vertex y_k, leading to an exponential complexity of this easy-to-describe algorithm. The first recursive call $\mathcal{H}_{k-1} \cup \{u_k, v_k, y_k\}$ indeed identifies the winning sets for this subgame for both players. Moreover, for Steven, the strategies that are winning in this subgame are in fact winning in $\mathcal{H}_k$ too. Unfortunately, the next recursive call promptly discards this information.

Proof of Lemma 8.1.1. Without loss of generality, we assume that n is even. The same claims hold for odd n. Consider the important families of subgames that we see in our recursive calls:

- $\mathcal{B}_n = \mathcal{H}_{n-1} \cup \{u_n, v_n\} = \mathcal{H}_n \setminus \{w_n, x_n, y_n\}$
- $\mathcal{C}_n = \mathcal{H}_{n-1} \cup \{u_n, v_n, x_n\} = \mathcal{H}_n \setminus \{w_n, x_n\}$

We will argue that each of these families take exponential time to solve due to the structure of the game.

Proposition 8.1.2. *The number of recursive calls needed to solve $\mathcal{C}_n$ is the number of recursive calls needed to solve $\mathcal{B}_n$. Moreover, the number of recursive calls to solve either subgame using Algorithm 7 is 2^n.*

Proof. We prove the above proposition by proving the following claim. If we denote the number of recursive calls to solve $\mathcal{C}_n$ and $\mathcal{B}_n$ with $C(n)$ and $B(n)$ respectively, then we also have the following simultaneous recurrence relation $C(n) = B(n) = C(n-1) + B(n-1)$ of the claim.

Claim 1. McNZ-Even *on the subgame $\mathcal{B}_n$ as well as $\mathcal{C}_n$ makes two recursive calls of* McNZ-Odd *on $\mathcal{B}_{n-1}$ and then $\mathcal{C}_{n-1}$.*

The top priority in both games is $n+1$. The Steven attractor to the vertices of priority $n+1$ is

- $\{u_n, v_n, y_n, w_{n-1}, x_{n-1}\}$ in $\mathcal{C}_n$ and
- $\{x_n, y_n, w_{n-1}, x_{n-1}\}$ in $\mathcal{B}_n$

The complement of the Audrey attractor to $n+1$ in both these games is exactly $\mathcal{C}_{n-1}$ and their first recursive call for both the games are the same.

This recursive call returns only $\{u_{n-1}, v_{n-1}\}$ as the winning sets for Steven in $\mathcal{C}_{n-1}$. The Steven attractor to $\{u_{n-1}, v_{n-1}\}$ is $\{u_{n-1}, v_{n-1}, w_{n-1}\}$ in both $\mathcal{C}_n$ and $\mathcal{B}_n$.

The complement of this Steven attractor set is

- $\mathcal{B}_{n-1} \cup \{u_n, v_n, y_n\}$ in $\mathcal{C}_n$ and
- $\mathcal{B}_{n-1} \cup \{x_n, y_n\}$ in $\mathcal{B}_n$

The top priorities in the above subgames are

- $\{u_n, v_n, y_n, w_{n-1}\}$ in $\mathcal{C}_n$, with Audrey's attractor being $\{u_n, v_n, y_n, w_{n-1}, x_{n-1}\}$ and
- $\{x_n, y_n, w_{n-1}\}$ in $\mathcal{B}_n$, with Audrey's attractor being $\{x_n, y_n, w_{n-1}, x_{n-1}\}$.

Their complements in the respective games is $\mathcal{B}_{n-1}$. □

We only need to make the following statement to conclude our proof.

Claim 2. McNZ-Even *on the subgame* $\mathcal{H}_n$ *makes two recursive calls of* McNZ-Odd *to the subgames* $\mathcal{C}_{n-1}$ *and* $\mathcal{B}_{n-1}$.

In fact, the complement of the Steven attractor to the top priority of $\mathcal{H}_n$ is the set $\mathcal{C}_n$, which is the first recursive call. The Audrey winning set in $\mathcal{C}_n$ is exactly the set $\{u_n, v_n\}$. The Audrey attractor to this two-element set in the entire game is $\{u_n, v_n, w_n\}$. Now we are left with the subgame $\mathcal{B}_n$, thus proving our claim. □

8.1.2 McNaughton-Zielonka algorithm with memory

McNaughton-Zielonka's wasteful behaviour in the example family of games above prompts us to ask if there is some way we can utilise the progress we make in the first recursive call of these symmetric algorithms to provide a head-start for the following recursive calls. Several enhancements of McNaughton-Zielonka exist, all of which attempt to utilise information from recursive calls. Some notable ones include the priority promotion algorithms and its variants [BDM18, BDM$^+$21] along with the Tangle learning algorithm by van Dijk [vD18], which all perform well in practice. The key idea behind these algorithms was to identify quasi-dominions (which are subgames where a player wins from all vertices if the play stays within this subgame) in their recursive calls and to increase these quasi-dominions obtained so far carefully. Another approach was to remember the strategy obtained in the recursive call. A recent work by Lapauw, Bruynooghe, and Denecker [LBD20] shows that by remembering and modifying the strategies obtained recursively in a calculated

manner, faster practical performances can be obtained for several benchmarks. But all of these have worst-case exponential running time comparable to McNaughton-Zielonka, as these heuristics do not lead to a provable increase in the worst-case running time.

We show that our idea of enhancing an algorithm with decompositions can be adapted to work for recursive, symmetric, attractor-based algorithms. This turns out to reduce the worst-case runtime complexity to a square-root of the McNaughton-Zielonka algorithm. We make use of structural information obtained from recursive calls to significantly reduce the worst-case overall size of the tree of recursive calls of the algorithm as compared to other symmetric attractor-based approaches. We show how witnesses for both players from recursive calls on subgames can be meaningfully used to reduce the sizes of subgames on which further recursive calls are made, even if their key properties are damaged by the removal of some vertices from subgames on which they were computed. In contrast, other symmetric attractor-based algorithms are wasteful by routinely discarding witnesses for one of the players that are computed in recursive calls; in the worst case, this results in repeatedly solving large subgames from scratch.

The procedure McNZFast-Even in Algorithm 8 works using decompositions $\mathcal{D}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}}$ on a subgame $\mathcal{G}$. Both Audrey's and Steven's decompositions are being maintained, and they both need to be modified based on the attractors computed. To distinguish between the modification of each of these decompositions, we highlight the changes made to the Steven decomposition in blue and the changes made to the Audrey decomposition in pink. We also observe that the highlighted part indicates exactly how our enhancement deviates from the attractor-decomposition producing version of McNaughton-Zielonka (Algorithm 7). Indeed, removing the highlighted parts gives us the McNaughton-Zielonka algorithm.

We assume that the algorithm takes as input an (n, h)-small parity game $\mathcal{G}$, the priority h, the nodes ϵ in tree $\mathcal{T}^{\text{Even}}$ with even level h, and the node ω in tree $\mathcal{T}^{\text{Odd}}$ with odd level $h+1$. If for one of the players, the decomposition computed so far already forms an attractor decomposition, then the algorithm stops and returns the corresponding set, since having an attractor decomposition implies that we have a witness of winning for that player in the current subgame. On the other hand, if both players only have decompositions that are not attractor decompositions of the current subset of vertices, our algorithm computes the Steven attractor to the top priority in the subgame $\mathcal{G}_i$, closely mirroring the McNaughton-Zielonka algorithm (Algorithm 7). However, our procedure deviates from McNaughton-Zielonka here by modifying the part $T^{\epsilon}_{\text{Even}}$ of the Steven decomposition to update this information.

Algorithm 8 McNaughton and Zielonka Algorithm with memory

Input: A parity games $\mathcal{G}$ with highest priority h and nodes ϵ in the even tree $\mathcal{T}^{\text{Even}}$ whose even level is h and ω in the odd tree $\mathcal{T}^{\text{Odd}}$ whose odd level is $h+1$.

Output: The set of winning vertices for Steven.

1: **procedure** McNZFast-Even($\mathcal{G}, h, \epsilon, \omega$)
2: **if** $\mathcal{D}^{\epsilon}_{\text{Even}}$ restricted to $\mathcal{G}$ is an attractor decomposition **then**
3: Set $\left(S^{\omega}_{\text{Odd}}, V(\mathcal{G})\right)$
4: **return** $V(\mathcal{G})$
5: **else if** $\mathcal{D}^{\omega}_{\text{Odd}}$ restricted to $\mathcal{G}$ is an attractor decomposition **then**
6: Set $\left(S^{\epsilon}_{\text{Even}}, V(\mathcal{G})\right)$
7: **return** $\varnothing$
8: **else**
9: $\mathcal{G}_1 \leftarrow \mathcal{G}$; $i = 0$
10: **repeat**
11: $i \leftarrow i + 1$
12: $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$
13: $T_i \leftarrow$Steven attractor to H_i in $\mathcal{G}_i$
14: $R_i \leftarrow T^{\epsilon}_{\text{Even}} \setminus (T_i \cup H_i)$
15: $\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$
16: $S_i \leftarrow T_i \cap \left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$
17: Set $\left(S^{\omega \odot i}_{\text{Odd}}, S_i\right)$
18: $\mathcal{G}'_i \leftarrow \left(\mathcal{G}_i \cap \left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]\right)$
19: $U'_i \leftarrow$ McNZFast-Odd($\mathcal{G}'_i, h-1, \epsilon, \omega \odot i$)
20: $S'_i \leftarrow$ Audrey attractor to U'_i in $\mathcal{G}_i$.
21: $R'_i \leftarrow S^{\omega \odot i}_{\text{Odd}} \setminus S'_i$
22: $\text{Move}_{\mathcal{D}_{\text{Odd}}}\left(R'_i\right)$
23: $Q_i \leftarrow S'_i \cap \left[\mathcal{D}^{\epsilon}_{\text{Even}}\right]$
24: Set $\left(S^{\epsilon}_{\text{Even}}, Q_i\right)$
25: $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus S'_i$
26: **until** $\mathcal{D}^{\epsilon}_{\text{Even}}$ restricted to $\mathcal{G}_{i+1}$ is an attractor decomposition
27: **return** $V(\mathcal{G}_i)$
28: **end if**
29: **end procedure**

This is done in the lines 14 and 15, where the set R_i is first identified as the set of vertices that are no longer in the attractor to the top priority and later the subroutine $\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$ is called on such an R_i. This subroutine modifies the decomposition $\mathcal{D}_{\text{Even}}$ such that the vertices in R_i are assigned to parts in the decomposition labelled by values larger than ϵ. The Move subroutine was defined in detail in the previous chapter.

The lines highlighted in blue result in the current Steven decomposition being modified in such a way that the top-set $T^{\epsilon}_{\text{Even}}$ now contains exactly the vertices $T_i \setminus H_i$, from which Steven has a reachability strategy to the set of vertices of highest priority H_i in the subgame $\mathcal{G}_i$.

Next, the lines highlighted in pink result in the modification of Audrey's decomposition. The algorithm first computes the vertices S_i that are in the intersection of T_i (the Steven attractor to H_i) and $\left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$. Steven has a strategy to escape from S_i in the subgame $\left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$ using the attractor strategy in T_i and visit an even vertex of priority h. To ensure that $\left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$ is a trap for Steven, using the subroutine Set, the vertices in S_i are removed from the parts of the decomposition that are contained in $\left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$ and later reassigned to the parts containing vertices in S_i in Audrey's decomposition. The vertices in S_i are now reassigned instead to the part $S^{\omega \odot i}_{\text{Odd}}$. This subroutine Set is more nuanced, but for the moment, it is best thought of as being reassigned to the part represented by $S^{\omega \odot i}_{\text{Odd}}$. We provide a more technical overview later on for $S^{\omega \odot i}_{\text{Odd}}$.

In the original McNaughton-Zielonka algorithm, the next recursive call works on the complement of the Steven attractor set T_i. The complement of a Steven attractor set forms a trap for him. However, in the next recursive call at one level lower, our algorithm considers the subgame $\mathcal{G}'_i$ of $\mathcal{G}_i$ induced by $\left[\mathcal{D}^{\omega \odot i}_{\text{Odd}}\right]$. We then perform an odd-level recursive call by calling the procedure McNZFast-Odd on $\mathcal{G}'_i$, which is a trap for Steven. After Audrey's recursive call returns her winning set U'_i in the subgame $\mathcal{G}'_i$, the algorithm computes the Audrey attractor S'_i to the set U'_i. Since this attractor S'_i is the set of vertices from which Audrey has a strategy to reach U'_i, we adjust Audrey's decomposition by calling the subroutine Move on vertices R'_i that are in $S^{\omega \odot i}_{\text{Odd}}$ but not in S'_i. This is captured in line 22 which calls the subroutine Move on R'_i, but this time for Audrey's decomposition. Finally, the procedure computes the set of vertices Q_i in both the set S'_i and the set $\left[\mathcal{D}^{\epsilon}_{\text{Even}}\right]$. The subroutine Set is called which relocates Q_i to the part $S^{\epsilon}_{\text{Even}}$.

We initialise these decompositions, similar to our asymmetric algorithm, with the smallest decomposition for both players.

Remark 5. *The subroutine* $\text{Set}\left(S^{\eta}_{\mathcal{D}}, S\right)$ *takes as input a part from a decomposition*

$\mathcal{D}$. *Intuitively, the subroutine modifies it in such a way that all vertices in* S *are now assigned to the part* $S^{\eta}_{\mathcal{D}}$. *It can be performed by the following operations:*

- $\left[\mathcal{D}^{\eta}\right] \leftarrow \left[\mathcal{D}^{\eta}\right] \ominus S$
- $S^{\eta}_{\mathcal{D}} \leftarrow S^{\eta}_{\mathcal{D}} \cup S$

Recall that the operator $\ominus$ *was defined in the previous chapter in page 106. The time taken by such operations is near-linear time.*

However, we remark that any subroutine that modifies the decomposition to a decomposition $\mathcal{D}'$ *that it is*

1. *at least as large as the original decomposition* $\mathcal{D}$,
2. *at least as large as the decomposition* $\mathcal{E}$ *which has* $S^{\eta}_{\mathcal{E}} = S$ *and all other parts in the partition* $\mathcal{E}$ *to empty, and*
3. *at most as large as the smallest attractor decomposition larger than* $\mathcal{D}$

serves our purposes in terms of proving correctness.

Correctness and Runtime. Any function Move as defined in the previous chapter and Set as in Remark 5 provide the required correctness and runtime that we claim in the theorem below. We only state the following theorem, which guarantees correctness and a quadratic improvement in the runtime, but prove it towards the end of the chapter.

Theorem 8.1.3. *Let* $\mathcal{G}$ *be a* (n, h)*-parity game and let* ϵ *and* ω *be nodes that have even level and odd level* h *and* $h+1$ *in the two* $\mathbb{N}$*-labelled* n*-ary trees* $\mathcal{T}^{\text{Even}}$ *and* $\mathcal{T}^{\text{Odd}}$ *respectively. Procedure* McNZFast-Even$(\mathcal{G}, h, \epsilon, \omega)$*, initialised with the smallest* $\mathcal{T}^{\text{Even}}$*-decomposition of* $\mathcal{G}$ *for Steven and the smallest* $\mathcal{T}^{\text{Odd}}$*-decomposition of* $\mathcal{G}$ *for Audrey, terminates with the smallest Steven* $\mathcal{T}^{\text{Even}}$*-attractor decomposition for the Steven dominion and the smallest Audrey* $\mathcal{T}^{\text{Odd}}$*-attractor decomposition for the Audrey dominon in* $\mathcal{G}$.

The following lemma highlights the improved runtime of our algorithm by a quadratic improvement we gain compared to McNaughton-Zielonka, and comparable to the small progress measure algorithm by Jurdziński [Jur00].

Lemma 8.1.4. *For a* (n, h)*-small parity game* $\mathcal{G}$*, the number of recursive calls by either* McNZFast-Even *or* McNZFast-Odd *is at most the product of a polynomial in* n *and* $\mathcal{O}\left(\frac{n}{h}\right)^{\lceil h/2 \rceil}$.

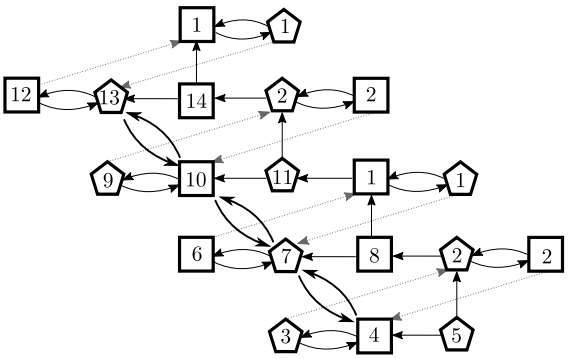


Figure 8.2: The game $\mathcal{F}_4$

8.1.3 Analysis on example families of games

We will demonstrate the algorithm on two examples in this subsection to understand the efficiency of our algorithm. These two examples are going to be the family of games $\mathcal{H}_k$ introduced in Section 8.1 and the family of games, we will call $\mathcal{F}_k$, which was introduced by Friedmann [Fri11].

We recall that in his work, Friedmann provided a family of games on which McNaughton-Zielonka takes exponential time [Fri11]. We will call this family of examples $\mathcal{F}_k$ for $k \in \mathbb{N}$.

We will again describe this game $\mathcal{F}_n$ in 'layers' as done for the previous hard example. The game $\mathcal{F}_n$ consists of $n+1$ layers $L_0, , L_1, \ldots, L_n$, each consisting of at most 5 vertices.

- L_0 consists of 3 vertices a_0, b_0 and c_0.
- For $i \in \{1, \ldots, n-1\}$, we define L_i to be $\{a_i, b_i, c_i, d_i, e_i\}$.
- Finally, $L_n = \{d_n, e_n\}$.

The priorities of the vertices a_i, b_i, and c_i are $3i+3$, $3i+4$, and $3i+5$ respectively. The priorities of d_i and e_i are both 2 if i is odd and 1 if i is even. The ownership in a layer alternates. Vertices a_i, c_i and e_i are owned by Audrey when i is even and owned by Steven otherwise. The vertices b_i and d_i are owned by Audrey when i is odd and by Steven otherwise. The edges within a layer are $(a_i, b_i), (b_i, a_i), (c_i, b_i), (d_i, c_i), (d_i, e_i)$, and (e_i, d_i). Between the layers, we define the following edges $(b_i, b_{i+1}), (b_{i+1}, b_i), (a_i, d_{i+1})$, (e_{i+1}, b_i), and (c_i, d_{i+1}), for $i \in \{0, \ldots, n-1\}$. These edges however are defined only when both the incoming and outgoing vertices corresponding to it exist. The game $\mathcal{F}_n$ is won by Audrey for even values of n and is won by Steven otherwise. We have drawn $\mathcal{F}_4$ in Figure 8.2,

and this game is won by Audrey. The winning strategy for the player that wins the game is to either 'go left' or 'go up'.

The exponential behaviour is due to the careless disposal of information obtained in recursive calls. In the next two lemmas, we show that our algorithm solves Friedmann games as well as our family of games $\mathcal{H}_k$ in polynomial time.

Lemma 8.1.5. *Algorithm 8, when initialised with the trivial decomposition for both players, solves the family of games $\mathcal{H}_n$ in time that is polynomial in n.*

Lemma 8.1.6. *Algorithm 8, when initialised with the trivial decomposition for both players, solves the Friedmann family of games $\mathcal{F}_n$ [Fri11] in time that is polynomial in n.*

The key idea behind both of these proofs is that the algorithm records a sufficient amount of progress by maintaining a decomposition in its preliminary recursive calls. In future recursive calls, when this decomposition is our starting point, the algorithm does at most polynomial amount of processing in the above family of games. This phenomenon is highlighted in the scenarios listed below.

1. Steven or Audrey already has a winning strategy in the form of an attractor decomposition from a previous recursive call (as in the proof of Lemma 8.1.5 as well as Lemma 8.1.6).

2. Audrey's decomposition is robust enough to conclude quickly that the game is losing for Audrey, and therefore winning for Steven (as in the proofs of Lemma 8.1.5 and Lemma 8.1.6).

3. Subsequent recursive calls are made on significantly smaller subgames due to the structure of the available decompositions. Some of these recursive calls might even turn out to be empty (as in the proof of Lemma 8.1.5).

The exact details of these proof requires us to identify specific subgames that the recursive calls are made on and the details of the proof are available below. Our proofs, which show that these family of games are solvable in polynomial time, rely on induction. Thus, in turn, hinges on the regularity of the subgames in recursive calls.

While these proofs are tailored to exploit the regular structures within the subgames and establish the polynomial termination of these games, it is essential to note that our algorithm's strength transcends being a mere tailored technique reliant on game-specific properties. We underscore this by showing that our algorithm's

efficiency is not confined to a specialized context; indeed, it operates with a worst-case time complexity that is a square root of that of the McNaughton-Zielonka algorithm.

Fast termination of Algorithm 8 on games $\mathcal{H}_n$ and $\mathcal{F}_n$

For the proofs, we drop the subscripts Even and Odd from decompositions and their respective sets to simplify notation. The decompositions will be identified by their node in the tree, ϵ denoting nodes in the Steven tree and ω denoting nodes in Audrey's tree.

Proof of fast termination of Algorithm 8 on the family of games $\mathcal{H}_n$ We prove this by showing that the decomposition maintained after one recursive call ensures polynomial termination of the other recursive call.

Proof (sketch) of Lemma 8.1.5. Recall the families of subgames from the proof of Lemma 8.1.1, showing exponential runtime of the games defined where for all k, we define $\mathcal{C}_k = \mathcal{H}_k \setminus \{w_k, x_k\}$.

We will find the number of recursive calls to solve $\mathcal{C}_{k+1}$ in this proof instead. This suffices because the complement of the attractor to the top priority in $\mathcal{C}_{k+1}$ as well as $\mathcal{H}_k$, remain the same for these two games on running the two algorithms.

If $T(k)$ denotes the time taken to solve $\mathcal{C}_{k+1}$, then we show that T satisfies the recurrence relation for a fixed constant c

$$T(k+1) \leqslant T(k) + c \cdot k^c.$$

From the above recurrence, we can show that $T(k+1)$ is bounded by a polynomial. Now all that remains is to show that the algorithm on $\mathcal{C}_{k+1}$ in fact produces the above recurrence. Without loss of generality, assume k is even.

First iteration. The vertices of priority $k+2$ in $\mathcal{C}_{k+1}$ are exactly the vertices in the set $\{u_{k+1}, v_{k+1}, y_{k+1}, w_k\}$. The algorithm computes the Steven attractor to this set of vertices as it is the set of vertices that have the highest even priority. The Steven attractor to it consists of the above set along with vertex x_k. These vertices are added to H^ϵ and T^ϵ respectively, where ϵ is the root of the Steven tree. The complement of these vertices is exactly $\mathcal{C}_k$. A recursive call is then made to the rest of the game $\mathcal{C}_k$ with decompositions that are the smallest decompositions (with respect to the order introduced on the decompositions by the decomposition labellings, as defined in Chapter 7) for both players, from where we get the first

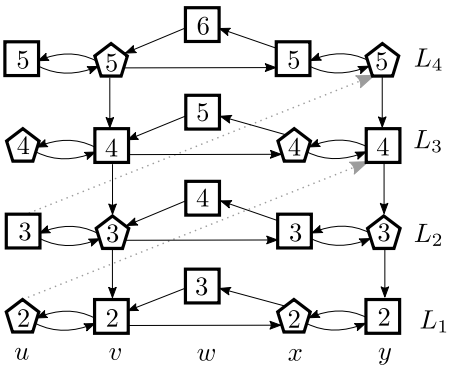


Figure 8.3: The game $\mathcal{H}_4$ recalled

term of our recurrence relation. The recursive sub call returns the attractor decompositions of $\mathcal{C}_k$.

Note that the subgame consisting exactly of the vertices $\{u_k, v_k\}$ is winning for Audrey in $\mathcal{C}_k$, while the rest are winning for Steven. Therefore, there is an attractor decomposition of the game $\mathcal{C}_k \setminus \{u_k, v_k\} = \mathcal{H}_{k-1} \cup \{y_k\}$. This game is winning for Steven and therefore has an attractor decomposition for Steven. The strategy for Steven is to use the same edges from the odd layers, which was winning for Steven in $\mathcal{H}_k$. For even layers, Steven's strategy in this subgame would be to use the edges (u_i, y_{i+1}) and (x_i, w_i) for even values of i.

We show that the algorithm returns the smallest attractor decomposition of the subgame $\mathcal{H}_{k-1} \cup \{y_k\}$. We will now define the attractor decomposition of the subgame $\mathcal{H}_{k-1} \cup \{y_k\}$, which will be returned by our first recursive call.

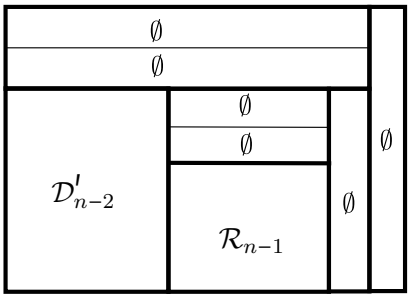
To avoid cumbersome notation to define our attractor decomposition for the subgame $\mathcal{H}_{k-1} \cup \{y_k\}$, we define the attractor decomposition of $\mathcal{H}_{n-1} \cup \{y_n\}$ for any even n, and denote it using $\mathcal{A}_n^{\epsilon}$. The tree with respect to which the attractor decomposition $\mathcal{A}_n^{\epsilon}$ and $\mathcal{A}^{\omega}$ is defined is not a complete tree; however, we remark that this tree can be embedded into the complete tree, and therefore we can extend the decomposition into one with respect to a complete tree. We will show how this decomposition can be obtained in a step-by-step manner.

The attractor decomposition $\mathcal{A}_n^{\omega}$ is defined as

$$\left\langle \varnothing, \varnothing, \left(\mathcal{D}'_{n-2}, \langle \varnothing, \varnothing, (\mathcal{R}_{n-1}), \varnothing \rangle \right), \varnothing \right\rangle$$

where $\mathcal{D}'_{n-2}$ is an $(\omega_1, \mathcal{T})$-decomposition and $\mathcal{R}_{n-1}$ is an $(\omega_{21}, \mathcal{T})$-decomposition. We define these decompositions inductively as follows. The $(\omega_1, \mathcal{T})$-decomposition $\mathcal{D}'_{n-2}$ is the same as another decomposition $\mathcal{D}_n$ but with its side-set defined as

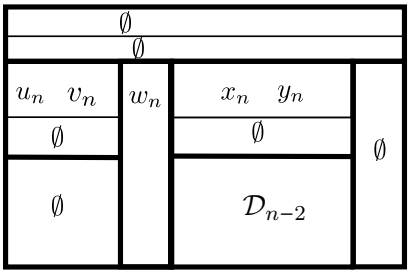
$S_{\mathcal{A}}^{\omega_1} \coloneqq \{y_{n+1}, u_{n+1}\}$. We pictorially represent $\mathcal{A}_n^{\omega}$ for better readability.



We now define the attractor decompositions $\mathcal{D}_{n-2}$ and $\mathcal{R}_{n-1}$ recursively. We define $\mathcal{D}_n$ and $\mathcal{R}_{n-1}$ inductively as drawn below. We say $\mathcal{D}_n$ is the decomposition

$$\langle \varnothing, \varnothing, (\langle \{u_n, v_n\}, \varnothing, \varnothing \rangle, \langle \{x_n, y_n\}, \varnothing, (\mathcal{D}_{n-2}), \varnothing \rangle) \rangle .$$

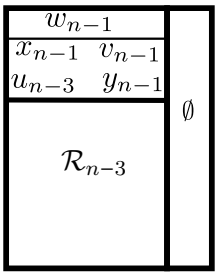
For improved readability, we draw the decomposition as follows.



We define $\mathcal{R}_{n-1}$ as

$$\langle \{w_{n-1}\}, \{x_{n-1}, v_{n-1}, u_{n-1}, y_{n-1}\}, (\mathcal{R}_{n-3}), \varnothing \rangle$$

and again draw this decomposition as follows.



.

Once $\mathcal{A}_{k-1}^{\epsilon}$ is returned, the Audrey attractor is computed to the two-vertex winning set $\{u_k, v_k\}$ from before, whose Audrey attractor in the subgame $\mathcal{C}_{k+1}$ turns

out to contain only the extra vertex w_k. Now, the respective Set and Move operations are performed, expelling the vertex w_k from T^ϵ, into S^{ϵ_1} of the even decomposition maintained by the algorithm, thus completing the first iteration of the outermost recursive call.

Second iteration. In the second iteration, there are no 'extra vertices' in the attractor to the set of top even priority vertices in the current subgame considered. This is because now the only top even priority vertices are u_{k+1}, v_{k+1} and y_{k+1}. The complement of it is the subgame $\mathcal{H}_k \cup \{x_k, y_k\}$. Therefore, its attractor turns out to be empty, so the vertex x_k, which was previously in T^ϵ instead, is now assigned to S^{ϵ_1}. We now argue that the odd-level recursive calls made terminate after only polynomial work. We do this by closely analysing what the odd recursive call does in each of these recursive calls.

For each child ϵ_i of ϵ, there is an odd recursive call with the root of Audrey's tree ω_2 and the Steven tree ϵ_i.

- On observing the labelling maintained by Steven after the previous recursive call, we can conclude that for ϵ_1, the number of vertices in the recursive call itself is constant.
- For ϵ_2, all the vertices in the scope already form an attractor decomposition, so the work done is polynomial.
- This leaves ϵ_3, but the set of vertices in the scope turns out to be exactly the subgame $\mathcal{O}_{k-1}$, which consists of only the odd layers of the game $\mathcal{H}_{k-1}$ (defined later), for which the algorithm takes polynomial time on this subgame from Proposition 8.1.7 below.

□

We will call the subgame of $\mathcal{H}_n$ restricted to the odd layers $\mathcal{O}_n$ and the even layers $\mathcal{E}_n$. Let us prove the following proposition, which will help us to arrive at a recurrence relation for the above lemma.

The decomposition $\mathcal{R}'_n$ in the following lemma is defined similarly to $\mathcal{R}_n$ defined previously, but with the top-set T^ϵ containing the vertex u_{n-1} additionally. The proof is by a simple induction.

Proposition 8.1.7. *For the family of games $\mathcal{O}_n$ such that n is odd, the procedure* McNZFast-Even *solves this family in polynomial time where the initial decomposition for Steven is $\mathcal{R}'_n$ and the initial decomposition for Audrey is the smallest decomposition of $\mathcal{O}_n$.*

Proof sketch. When $n = 3$, the statement is trivial.

For larger n, the algorithm first computes the Steven attractor $\{w_n, x_n\}$ to the set of vertices with the highest priority $\{w_n\}$. Following this, an odd recursive call is made, which computes an Audrey attractor to set of vertices $\{u_n, v_n, y_n\}$, which have the highest priority. When removing the set of vertices first in the Audrey attractor and then in the Steven attractor, we are left with a similar substructure of decomposition with vertices of $\mathcal{O}_{n-2}$. Since only linear time passes before making a recursive call to a game $\mathcal{O}_{n-1}$, the time taken in is polynomial. □

Proof of fast termination of the family $\mathcal{F}_n$. We consider the family of graphs $\mathcal{F}_n$. The crux of the argument is that we identify the decomposition that is obtained after the recursive call for this family and show that starting from this decomposition takes only polynomial time.

Let us try to answer a simple question: What does the smallest attractor decomposition of $\mathcal{F}_n$ look like?

Due to the symmetric nature of the family and the algorithm, it is enough to answer this question for an even-valued n. The analysis for odd-valued n is similar.

We denote by $\mathcal{A}_n^\omega$ the attractor decomposition defined below for Audrey of the game $\mathcal{F}_n$, where ω is the root of Audrey's tree at level $3n+3$. We will define $\mathcal{A}_n^\omega$ inductively, but before we do that we describe the tree $\mathcal{T}$ where $\mathcal{A}_n^\omega$ is an $(\omega, \mathcal{T})$-attractor decomposition. This is such that the root has an odd level of $3n+3$. This is because the highest priority in the game $\mathcal{F}_n$ is $3n+4$. Now, we describe the tree of odd level $3n+3$, and say $\mathcal{T}_3$ is the tree with one leaf of odd level 3 and written as the naturally labelled tree

$$\mathcal{T}_{3n+3} = \langle (1, \langle\rangle), (2, \langle (1, \langle (1, \mathcal{T}_{3n-3})\rangle)\rangle)\rangle\rangle\,.$$

Henceforth, we assume that we refer to $(\omega, \mathcal{T}_{3n+3})$-decompositions, for the tree corresponding to one defined above.

If $n = 2$, then $\mathcal{A}_n^\omega$ is the $(\omega, \mathcal{T}_{3n+3})$-attractor decomposition consisting of $H^\omega = \{b_1\}$, $T^\omega = \{a_1, d_2, e_2\}$ and $S^\omega = \{c_1, d_1, e_1, a_0, b_0, c_0\}$.

For $n > 2$ and even valued n, we have the attractor decomposition $\mathcal{A}_n^\omega$ defined as follows.

- For the first child ω_1 of ω: $H^{\omega_1} = \{b_{n-1}\}$, $T^{\omega_1} = \{a_{n-1}, d_n, e_n\}$ and $S^{\omega_1} = \{c_{n-1}, d_{n-1}, a_{n-2}, c_{n-2}\}$.
- Let ω' be the first descendant of ω_2 at level $3n-3$. Note that this would be the first child of the first child of ω_2. If confirming to the notation of adding

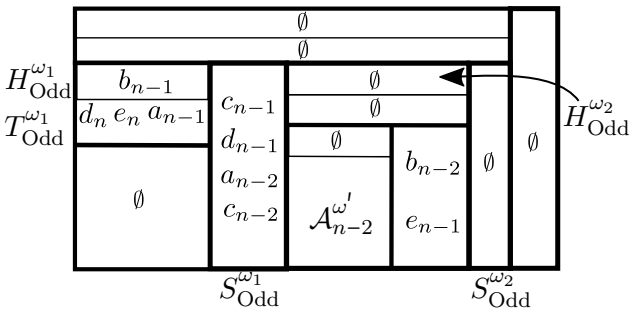


Figure 8.4: $\mathcal{A}^{\omega}_{n}$, attractor decomposition of the game $\mathcal{F}_n$

the order of children to the subscript, we would have called it ω_{211}, which is cumbersome. Hence, we refer to it as ω. However, we have for the first child of ω_2, denoted by ω_1, $S^{\omega_{21}} = \{b_{n-2}, e_{n-1}\}$ and $H^{\omega'} = T^{\omega'} = \varnothing$. Moreover $H^{\omega_2} = T^{\omega_2} = S^{\omega_2} = \varnothing$.

- The rest of the attractor decomposition at ω' is obtained by declaring the side attractor $S^{\omega'}_{\mathcal{A}}$ of the recursively defined $(\omega', \mathcal{T}_{3n+3})$-decomposition $\mathcal{A}^{\omega'}_{n-2}$ to be $\{b_{n-2}, d_{n-1}\}$.

A visual representation of this attractor decomposition is given in Figure 8.4 to aid in understanding the proof. We call $\mathcal{J}^{\omega}_{n}$ the decomposition obtained from $\mathcal{A}^{\omega}_{n}$ by restricting to the subgame $\mathcal{F}'_n = \mathcal{F}_n \setminus \{d_n, a_{n-1}, c_{n-1}\}$. This is obtained just by deleting $\{d_n, a_{n-1}, c_{n-1}\}$ from $\mathcal{A}^{\omega}_{n}$.

The following proposition shows that if $\mathcal{J}^{\omega}_{n}$ was the initial decomposition for Algorithm 8 on the subgame $\mathcal{F}'_n$, then the algorithm takes only polynomial time to return the winning set of $\mathcal{F}'_n$. Note that $\mathcal{F}'_n$ is winning from all vertices for Steven. and therefore the algorithm terminates with a Steven attractor decomposition for $\mathcal{F}'_n$.

Proposition 8.1.8. *For an even valued n (resp. odd), the procedure* McNZFast-Even *at level* $3n + 2$ *and* $3n + 1$ *for Steven and Audrey, when run on the game* $\mathcal{F}'_n$ *and initialised with the decomposition* $\mathcal{J}^{\omega}_{n}$ *for Audrey and the decomposition for Steven is the smallest decomposition, makes at most* $\mathcal{O}(n)$ *many recursive calls that are nonempty.*

One can similarly prove a proposition similar to the above for Steven, which we just state.

Proposition 8.1.9. *For an odd valued natural number n, the symmetric procedure* McNZFast-Odd *when run on the game* $\mathcal{F}'_n$*, initialised with the decomposition*

$\mathcal{J}_n^\epsilon$ for Steven (defined similarly) and the decomposition for Audrey is the smallest decomposition, makes at most $\mathcal{O}(n)$ many recursive calls that are non-empty.

We first provide the proof of Lemma 8.1.6 assuming the two previous propositions, which we prove later.

Proof of Lemma 8.1.6. Similar to the proof of Friedmann showing exponential termination, the algorithm computes the Steven attractor to highest even value: $3n+2$, which turns out to exactly be the vertex c_{n-1}. This is followed by the Audrey attractor to $3n+1$, which turns out to be the vertices $\{d_n, e_n, a_{n-1}, c_{n-1}\}$. After this, we are left with the vertices of $\mathcal{F}_{n-1}$ to work in the recursive call. Since n was even, $n-1$ is odd and $\mathcal{F}_{n-1}$ is winning for Steven.

This recursive call on $\mathcal{F}_{n-1}$ here computes the Steven decomposition of this set of vertices and returns it. Observe that this is the attractor decomposition $\mathcal{A}_{n-1}^\omega$ defined earlier.

Although all vertices are winning for Steven at this level, in the recursive call at the previous level $3n+1$ in the subgame $\mathcal{F}_n \setminus \{c_{n-2}\}$, the winning set for Audrey is $\{d_n, e_n, a_{n-1}, b_{n-1}\}$ and the winning set for Steven is $\mathcal{F}_{n-1}$.

After computing the attractor to Audrey's winning set $\{d_n, e_n, a_{n-1}, b_{n-1}\}$ in the subgame $\mathcal{F}_n$ and removing it from the game for the next set of vertices, what we are left to operate on is exactly $\mathcal{F}'_{n-1}$. The decomposition we have at hand restricted to these vertices is exactly the decomposition described above for $\mathcal{F}'_{n-1}$: $\mathcal{J}_n^{\hat{\omega}}$ where $\hat{\omega}$ has the level $3n$. From Proposition 8.1.9, we know that this terminates in polynomial time.

Let $T(n)$ denote the time taken to solve $\mathcal{F}_n$ when the Audrey and Steven decomposition are the initial Steven and Audrey decompositions possible for the vertices of $\mathcal{F}_n$, then we obtain the recursive relation:

$$T(n) = T(n-1) + F(n-2) + n^c$$

where $F(n)$ is the time taken to solve the game with the conditions given in Proposition 8.1.8. □

Proof of Proposition 8.1.8. We start with the decomposition $\mathcal{J}_n^\omega$ for Audrey and the smallest decomposition for Steven on the set of vertices of $\mathcal{F}'_n$. To prove the proposition, we show that the algorithm only makes $\mathcal{O}(1)$ many recursive calls and at most polynomial work on the decomposition both before and after making a recursive call on a subgame $\mathcal{F}'_{n-2}$ with the initial decomposition being $\mathcal{J}_{n-2}^\omega$ and the initial decomposition for Audrey. The above claim will show that $F(n)$ satisfies the

recurrence relation $F(n) = n^c + F(n-2)$, where $F(n)$ is the time taken for the algorithm on $\mathcal{F}'_n$ with the mentioned decompositions.

The first call is to McNZFast-Odd at level $3n+3$ at a node in the Audrey tree ω and level $3n+2$ for Steven with node ϵ although the highest priority in $\mathcal{F}'_n$ is $3n+1$ from vertex b_{n-1}.

For each child ω_i of ω, the algorithm computes the even attractor to vertices of priority $3n+2$. In each iteration, this is empty. Henceforth, to refer to the current decomposition maintained by the algorithm, we use $\mathcal{D}$ to refer and reserve $\mathcal{J}$ for the original one which the algorithm started with.

In the first iteration, the algorithm computes the Steven attractor to all vertices not in $\left[\mathcal{J}_n^{\omega_1}\right]$.

- Observe that $\left[\mathcal{J}_n^{\omega_1}\right] = \{e_n, b_{n-1}\}$ and the Steven attractor to the complement contains all the vertices of $\mathcal{F}'_n$. So, we get an empty recursive call.
- The vertices in $\left[\mathcal{J}_n^{\omega_1}\right]$ are moved to S^{ω_i} in the Audrey decomposition $\mathcal{D}^{\omega}$.
- The Audrey attractor to the empty set returned turns out to be empty. This all the vertices in S^{ω_i} are moved to $[\![\mathcal{D}_n^{\omega_2}]\!]$

We proceed to the next iteration with the same set of vertices, but the decomposition of Audrey is modified to include all vertices in $\left[\mathcal{D}_n^{\omega_2}\right]$. The algorithm is now at the next child of ω. For the next iteration, rooted at ω_2 observe that we have $H^{\omega_2} = b_{n-1}$ and T^{ϵ_2} is the set $\{e_n, c_{n-1}, d_{n-1}, a_{n-2}\}$ while the other partitions in the odd decomposition are the same as before in $\mathcal{D}_n^{\omega}$.

We enumerate the changes in the next few steps.

1. We start with the sets defined as $H^{\omega_2} = \{b_{n-1}\}$ and $T^{\omega_2} = \{e_n, c_{n-1}, d_{n-1}, a_{n-2}\}$ as mentioned above. However, the attractor to b_{n-1} is the set $\{b_{n-1}, e_n\}$, and hence in the next step, $T^{\omega_2} = \{e_n\}$. The algorithm expels the vertices $\{c_{n-1}, d_{n-1}, a_{n-2}\}$ to the first child of ω_2. Let us call this ω_2'. We have $H^{\omega_2'} = \{c_{n-2}\}$, given that it has priority $3n-1$. The rest of the vertices $\{d_{n-1}, a_{n-2}\}$ are now at $T^{\omega_2'}$.

2. An Steven recursive call is again made at level $3n$ to all vertices other than $\{b_{n-1}, e_n\}$, but since there are no vertices of that priority, this soon makes an Audrey recursive call at level $3n-1$. In this Audrey recursive call, is at Audrey's node ω_2'. In this recursive call, the Audrey attractor is computed to $H^{\omega_2'} = \{c_{n-2}\}$, which is just itself. Therefore, the rest of the items are moved to the first child of ω_2'.

3. Finally, another Steven recursive call is made; however, this only triggers an Audrey recursive call at level $3n-3$. We observe that at this level of the algorithm, we are only left with the vertices of $\mathcal{F}'_{n-2}$ and the decomposition restricted to $\mathcal{D}^{\omega}_{n-2}$. We see that there is not much change in the Steven decomposition and restricted to this level, the Steven decomposition is the smallest decomposition.

The work done above is polynomial before the recursive calls. At the end of the recursive call at level $3n-2$ for Steven and $3n-1$ for Audrey, we know that all the vertices of $\mathcal{F}_{n-2}$ along with $\{b_{n-2}, e_{n-1}, d_{n-1}, a_{n-2}, c_{n-2}\}$ are in the part $S^{\omega'}$ where ω' is the first descendant of ω_2 at level $3n-3$. Instead of writing it step by step, we conclude by saying that after 3 attractor computations for Audrey, all vertices are mapped to the side set of level $3n+2$, and all these vertices are returned at the end of the recursive call as winning for Steven. This gives us the required recurrence relation: $F(n) = n^c + F(n-2)$ for the time taken to solve the above. □

8.2 A symmetric attractor-based algorithm

The use of quasi-polynomial universal trees in the attractor-based algorithms of Parys [Par19, LPSW22], as well as that of Lehtinen, Schewe and Wojtczak [LSW19, LPSW22] was highlighted by Jurdziński and Morvan [JM20, JMT22]. The algorithm of Jurdziński and Morvan (Algorithm 1), when instantiated with specific universal trees, produces the algorithm of Parys [Par19] or Lehtinen, Schewe, and Wojtczak [LSW19]. A report on which universal trees correspond to which algorithm from their work has been summarised in Chapter 2 of the thesis. Therefore, we will deal with the more general attractor-based algorithm of Jurdziński and Morvan.

Although symmetric attractor-based algorithms are elegant, and efficient, the theoretical complexity of these algorithms does not match the complexity of state-of-the-art algorithms. The Jurdziński-Morvan algorithm takes time proportional to the product of size of the two trees that it is instantiated with. This is in contrast to the algorithms with state-of-the-art worst-case complexity [JL17], which takes time that is linear in the size of the tree. In fact, Lehtinen, Parys, Schewe and Wojtczak [LPSW22] emphasise that the complexity of quasi-polynomial attractor-based algorithms is almost the square of the complexity of other quasi-polynomial algorithms [JL17, FJdK+19]. In this section, we describe how our technique can be applied to the algorithm of Jurdziński and Morvan [JMT22], which in turn shows that the algorithm of Parys [Par19], as well as the algorithm of Lehtinen, Schewe and Wojtczak [LSW19, LPSW22] would benefit from our treatment.

Much like the approach adopted by Jurdziński and Morvan, where the recursive calls of the McNaughton-Zielonka algorithm were generalised to be dictated by two arbitrary trees, Algorithm 9, which generalises the decomposition-using version of McNaughton-Zielonka algorithm (Algorithm 8), also dictates its recursive calls within the framework of two arbitrary trees. Our algorithm, which uses two arbitrary trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$, contains two mutually recursive procedures Univ-Even-Fast and Univ-Odd-Fast. These procedures take as input a game $\mathcal{G}$, the highest priority h in the game, and two nodes ϵ and ω from $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$ respectively. The nodes ϵ and ω have level h and $h+1$ respectively in their trees for Steven's procedure Univ-Even-Fast and vice versa for Audrey's procedure Univ-Odd-Fast. The pseudocode for the Steven recursive call is given in Algorithm 9, whereas Audrey's is obtained by swapping the roles of the players. We emphasise that, except for the iterative loop, this algorithm is similar to the procedure in Algorithm 8.

Theorem F. *For a parity game $\mathcal{G}$ and two trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$, the procedure* Univ-Even-Fast *(resp.* Univ-Odd-Fast*) in Algorithm 9 (on page 140) takes time $n^{\mathcal{O}(1)} \cdot \mathcal{O}\big(\max\big(|\mathcal{T}^{\text{Odd}}|, |\mathcal{T}^{\text{Even}}|\big)\big)$ to identify a set of vertices that includes all Steven dominia of $\mathcal{G}$ with a $\mathcal{T}^{\text{Even}}$-attractor decomposition and does not intersect with any Audrey dominia with a $\mathcal{T}^{\text{Odd}}$-attractor decomposition.*

The rest of the section is dedicated to the proof of the above theorem. Similar to our proofs of correctness for the asymmetric version of the algorithm, the proof of correctness of our algorithms extensively uses the fact that our algorithm modifies this decomposition monotonically. The high-level arguments involved are that we increase the Steven and Audrey decompositions at each iteration. The correctness is based on the fact that we never increase the decompositions "too much". The running time complexity of our algorithm is based on the fact that it does not take "too long" before there is an increase in the value of decompositions at least in one of our decompositions. Let $\mathcal{A}_{\text{Even}}$ and $\mathcal{A}_{\text{Odd}}$ be the smallest attractor decomposition larger than the current decompositions $\mathcal{D}_{\text{Even}}$ and $\mathcal{D}_{\text{Odd}}$. We argue that the decompositions satisfy the invariant that $\mathcal{D}_{\text{Even}}$ and $\mathcal{D}_{\text{Odd}}$ are smaller than $\mathcal{A}_{\text{Even}}$ and $\mathcal{A}_{\text{Odd}}$, respectively, at each step. We, moreover, argue that between each increase to the decomposition, only polynomial time passes.

We emphasise that not all implementations of our algorithm would have the claimed running time, but we can achieve this with carefully designed data structures. One way to obtain a fast implementation would require a data structure that stores each decomposition as a decomposition labelling. This ensures that instead of keeping track of a partition with $O(|\mathcal{T}|)$ many parts, we can instead represent each node in the tree succinctly and store the node associated to the

Algorithm 9 A fast, symmetric, attractor-based algorithm

Input: A parity games $\mathcal{G}$ with highest priority h and nodes ϵ in the even tree $\mathcal{T}^{\text{Even}}$ whose even level is h and ω in the odd tree $\mathcal{T}^{\text{Odd}}$ whose odd level is $h+1$.

Output: A subset of vertices of $\mathcal{G}$ ▷ This subset corresponds to the winning vertices for Steven if $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$ are universal.

procedure UNIV-EVEN-FAST($\mathcal{G}, h, \epsilon, \omega$)
  if $\mathcal{D}^{\epsilon}_{\text{Even}}$ restricted to $\mathcal{G}$ is an attractor decomposition **then**
    Set $\left(S^{\omega}_{\text{Odd}}, V(\mathcal{G})\right)$
    return $V(\mathcal{G})$
  else if $\mathcal{D}^{\omega}_{\text{Odd}}$ restricted to $\mathcal{G}$ is an attractor decomposition **then**
    Set $\left(S^{\epsilon}_{\text{Even}}, V(\mathcal{G})\right)$
    return $\varnothing$
  else
    $\mathcal{G}_1 \leftarrow \mathcal{G}$
    for each ω_i among $\omega_1, \ldots, \omega_k$: children of ω **do**
      $H_i \leftarrow \pi^{-1}(h) \cap \mathcal{G}_i$
      $T_i \leftarrow$ Steven attractor to H_i in $\mathcal{G}_i$
      $R_i \leftarrow T^{\epsilon}_{\text{Even}} \setminus T_i$
      $\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$
      $S_i \leftarrow T_i \cap \left[\mathcal{D}^{\omega_i}_{\text{Odd}}\right]$
      Set $\left(S^{\omega_i}_{\text{Odd}}, S_i\right)$
      $\mathcal{G}'_i \leftarrow \left(\mathcal{G}_i \cap \left[\mathcal{D}^{\omega_i}_{\text{Odd}}\right]\right)$
      $U'_i \leftarrow$ UNIV-ODD-FAST($\mathcal{G}'_i, h-1, \epsilon, \omega_i$)
      $S'_i \leftarrow$ Audrey attractor to U'_i in $\mathcal{G}_i$
      $R'_i \leftarrow S^{\omega_i}_{\text{Odd}} \setminus S'_i$
      $\text{Move}_{\mathcal{D}_{\text{Odd}}}\left(R'_i\right)$
      $Q_i \leftarrow S'_i \cap \left[\mathcal{D}^{\epsilon}_{\text{Even}}\right]$
      Set $\left(S^{\epsilon}_{\text{Even}}, Q_i\right)$
      $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i \setminus S'_i$
    end for
  end if**return** $V(\mathcal{G}_{i+1})$
end procedure

leafy tree for each vertex. Since there are at most n vertices, this decomposition labelling would have a significantly smaller representation than storing a partition with as many parts as the size of either a quasi-polynomial or exponential sized tree. Moreover, using such a representation, we can ensure that recursive calls are never made on an empty set of vertices. This is achieved by only making a recursive call at a node of the tree when there is some vertex of the game whose decomposition labelling of this vertex is a descendant of this node.

We also argue that, in addition to making recursive subcalls, the algorithm only computes attractors and performs the respective Set and Move subroutines. Since these are relatively cheap operations, we are content with computing the number of recursive calls made by the algorithm to establish its running time up to a polynomial factor.

Lemma 8.2.1. *Let $\mathcal{G}$ be an (n, h)-small parity game where h is even, and let $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$ be two trees of with roots ϵ and ω respectively. Let $\mathcal{A}_{\text{Even}}$ be the smallest Steven $(\epsilon, \mathcal{T}^{\text{Even}})$-attractor decomposition of $\mathcal{G}$ and similarly $\mathcal{A}_{\text{Odd}}$, the smallest Audrey $(\omega, \mathcal{T}^{\text{Odd}})$-attractor decomposition. The procedure* Univ-Even-Fast$(\mathcal{G}, h, \epsilon, \omega)$*, with the decomposition being the initial Steven and Audrey decomposition of $\mathcal{G}$ outputs a set of vertices W such that $[\![\mathcal{A}^{\epsilon}_{\text{Even}}]\!] \subseteq W \subseteq V(\mathcal{G} \setminus [\![\mathcal{A}^{\omega}_{\text{Odd}}]\!])$.*

Proof. For this proof, we use the bijection between decompositions and decomposition labellings introduced in Chapter 6 and the order on decomposition introduced in Chapter 7. We lists the exact invariants required after each recursive subcall. If the decompositions at the beginning of a recursive call satisfy

1. $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$;
2. Steven has no strategy from any vertex in $\mathcal{G}$ to ensure the play proceeds to a vertex whose corresponding decomposition labelling is strictly smaller than ϵ in $\mathcal{L}\left(\mathcal{T}^{\text{Even}}\right)$ without visiting a vertex larger than ϵ^S
3. Audrey has no strategy from any vertex in $\mathcal{G}$ to ensure that the play proceeds to a vertex whose Audrey decomposition-labelling is strictly smaller than ω^T in $\mathcal{L}\left(\mathcal{T}^{\text{Odd}}\right)$, without visiting a vertex of with labelling ω^S.
4. $V(\mathcal{G}) = \left[\mathcal{D}^{\epsilon}_{\text{Even}}\right] \cap \left[\mathcal{D}^{\omega}_{\text{Odd}}\right]$.

then procedure Univ-Even-Fast described in Algorithm 9 on input $(\mathcal{G}, h, \epsilon, \omega)$ terminates with decompositions $\mathcal{E}^{\epsilon}_{\text{Even}}$ and $\mathcal{E}^{\omega}_{\text{Odd}}$:

(a) $\mathcal{E}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{E}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$;

(b) for each vertex $v \in V(\mathcal{G})$, at the end of the algorithm, either v belongs to $\left[\mathcal{E}^{\omega}_{\text{Odd}}\right]$ or $\left[\mathcal{E}^{\epsilon}_{\text{Even}}\right]$;

(c) the set of vertices returned is exactly $\left[\mathcal{E}^{\epsilon}_{\text{Even}}\right]$.

Moreover, if the two trees are complete n-ary trees, $\mathcal{E}^{\epsilon}_{\text{Even}} = \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{E}^{\omega}_{\text{Odd}} = \mathcal{A}^{\omega}_{\text{Odd}}$.

We start with two decompositions for vertices in $\mathcal{G}$: $\mathcal{D}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}}$, both smaller than the attractor decompositions $\mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{A}^{\omega}_{\text{Odd}}$ respectively. We show that each step of the algorithm modifies the decompositions $\mathcal{D}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\epsilon}_{\text{Odd}}$ in an inflationary manner whilst maintaining the invariant $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$.

We first deal with the if-else-if conditions in the algorithm that determine if the decomposition is already an attractor decomposition for either for the players.

$\mathcal{D}^{\epsilon}_{\text{Even}}$ is an attractor decomposition of $\mathcal{G}$ for Steven. Any play that stays within $\mathcal{G}$ is winning for Steven as the decomposition $\mathcal{D}^{\epsilon}_{\text{Even}}$ is a witness that Steven can win the game. So, Audrey can win if and only if the game exits the subgame of $\mathcal{G}$ in the larger game. If she remains in $\mathcal{G}$, she loses. If not, from Item 3, Steven has a strategy to ensure that Audrey visits a vertex mapped to ω^S or larger. Therefore, from the dual of Proposition 7.2.6, we know that $\left[\mathcal{A}^{\omega}_{\text{Odd}}\right]$ does not have any vertices in $\mathcal{G}$.

The algorithm moves all these vertices to S^{ω}_{Odd}. As there are no dominions for Audrey in $\mathcal{G}$, the smallest attractor decomposition larger than $\mathcal{D}^{\omega}_{\text{Odd}}$ would have to have all vertices at a position larger than ω^S. This ensures $\mathcal{E}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$. For each vertex $v \in V(\mathcal{G})$, at the end of the algorithm, all vertices v belong to $\left[\mathcal{D}^{\epsilon}_{\text{out}}\right]$, satisfying the second condition.

$\mathcal{D}^{\omega}_{\text{Odd}}$ is an attractor decomposition of $\mathcal{G}$ for Audrey. The arguments are similar to those above.

Both $\mathcal{D}^{\omega}_{\text{Odd}}$ and $\mathcal{D}^{\epsilon}_{\text{Even}}$ are not attractor decompositions. Here, we show that after the i^{th} iteration of Univ-Even-Fast,

- the invariant $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$ is preserved,
- each vertex in $\mathcal{G}_{i+1}$ is mapped by the current decomposition-labelling to a position that is strictly larger than ${\omega_i}^S$ by the Audrey decomposition-labelling obtained from $\mathcal{D}_{\text{Odd}}$.

We use subscripts to identify which parts of decomposition or attractor decomposition we are dealing with. For example, we use sets $H^{\epsilon}_{\mathcal{A}}, T^{\epsilon}_{\mathcal{A}}$, and $S^{\epsilon}_{\mathcal{A}}$ to refer to the sets in the smallest attractor decomposition $\mathcal{A}_{\text{Even}}$, and similarly $H^{\epsilon}_{\mathcal{D}}, T^{\epsilon}_{\mathcal{D}}$, and $S^{\epsilon}_{\mathcal{D}}$ for the decomposition $\mathcal{D}^{\epsilon}_{\text{Even}}$ and so on.

$\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$. The algorithm computes the attractor T_i to the set of vertices H_i containing the highest priority vertices and performs $\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$.

Since $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$, we know that for any element η in the leafy tree, the set of vertices mapped to a position smaller than η by the decomposition labelling corresponding to $\mathcal{D}^{\eta}$ is a superset of all vertices mapped to a position smaller than η by $\mathcal{A}^{\eta}$. The smallest element in the even decomposition is ϵ, and this gives us $\mathcal{G} \cap (H^{\epsilon}_{\mathcal{A}} \cup T^{\epsilon}_{\mathcal{A}}) \subseteq H_i \cup T_i$. The operations performed involve first identifying the set of vertices $R_i = T^{\epsilon}_{\text{Even}} \setminus T_i$. From any of these vertices in R_i, Steven does not have a strategy to reach H_i, which is a superset of $H^{\epsilon}_{\mathcal{A}}$. Therefore, we can conclude that from the vertices in R_i, Steven does not have a strategy to visit $H^{\epsilon}_{\mathcal{A}}$, and thus none of the vertices in R_i belongs to the top attractor set $T^{\epsilon}_{\mathcal{A}}$ in the attractor decomposition. Since each vertex in R_i is not present in $T^{\epsilon}_{\mathcal{A}}$, the subroutine Move increases the decomposition minimally whilst still staying below the least attractor decomposition.

$\text{Set}\left(S^{\omega_i}_{\text{Odd}}, S_i\right)$. Observe that S_i is the set of vertices in which Steven has a strategy to visit the set of vertices whose labellings in $\mathcal{D}^{\omega_i}$ currently at least as large as ${\omega_i}^S$. From Proposition 7.2.6, we can deduce that all vertices in S_i at positions (with respect to the current decomposition labelling) that are strictly smaller than ${\omega_i}^S$ have an empty intersection with $\left[\mathcal{A}^{\omega_i}_{\text{Odd}}\right]$. This turns out to exactly be the set of vertices that are re-labelled after the subroutine $\text{Set}\left(S^{\omega_i}_{\text{Odd}}, S_i\right)$ with a value larger than ${\omega_i}^S$ (the side attractor node of ω_i).

The recursive subcall on $\mathcal{G}'_i$. We outsource this task to induction. First, we observe that all the requirements are satisfied for the induction hypothesis.

1. After the recursive subcall on ϵ and ω_i, the decompositions are bounded above by the respective attractor decompositions.

2. Audrey has no strategy from any vertex in $\mathcal{G}'_i$ to ensure the play proceeds to a vertex whose corresponding decomposition labelling is strictly smaller than ω_i in $\mathcal{L}(\mathcal{T}^{\text{Odd}})$ without visiting a vertex larger than ${\omega_i}^S$. This follows because of the inductive invariants together with the arguments above for $\text{Set}\left(S^{\omega_i}_{\text{Odd}}, S_i\right)$.

3. Steven has no strategy from any vertex in $\mathcal{G}_i'$ to ensure that the play proceeds to a vertex whose corresponding decomposition labelling is strictly smaller than ${\epsilon^T}$ in $\mathcal{L}(\mathcal{T}^{\text{Odd}})$ without visiting a vertex larger than ${\epsilon^S}$. This is because of the line $\text{Move}_{\mathcal{D}_{\text{Even}}}(R_i)$.

4. Moreover, all vertices in $\mathcal{G}_i'$ are at parts such that either the Steven or Audrey decomposition (labelling) maps these vertices to a point at least as large as ${\epsilon^S}$ or ${\omega_i}^S$, respectively.

$\text{Move}_{\mathcal{D}_{\text{Odd}}}\left(R_i'\right)$**.** The vertices in $\mathcal{G}_i$ in $\left[\mathcal{D}^{\omega_i}\right]$ are exactly U_i'. Hence, any vertex in S^{ω_i} but not in S_i' has no attractor strategy to S^{ω_i}. Using arguments similar to the previous Move operation, we conclude that they preserve the invariants $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}_{\text{Odd}}$.

$\text{Set}\left(S^{\epsilon}_{\text{Even}}, Q_i\right)$**.** S_i' is the Audrey attractor to the set U_i'. All vertices in U_i', in the Steven decomposition are at a position at least as large as ${\epsilon^S}$. This is from the second invariant, combined with the observation that U_i' consists of exactly those vertices in $\left[\mathcal{D}^{\omega_i}_{\text{Odd}}\right]$. The algorithm computes Q_i which consists of all vertices in $\left[\mathcal{D}^{\epsilon}_{\text{Even}}\right]$ that have an Audrey attractor strategy to U_i'. From Proposition 7.2.6 in Chapter 7, we can conclude that the decomposition can be changed such that these vertices are now moved to a position that is at least the side attractor of ϵ, to the set $S^{\epsilon}_{\mathcal{D}}$. Such a change would preserve the invariants that we started with and would ensure that our modified decomposition is still smaller than the attractor decompositions.

Moreover, we are left exactly with vertices that are not mapped to values larger than ${\epsilon^S}$

After the k^{th} iteration. After the k^{th} iteration, we have shown that

1. the invariant $\mathcal{D}^{\epsilon}_{\text{Even}} \sqsubseteq \mathcal{A}^{\epsilon}$ and $\mathcal{D}^{\omega}_{\text{Odd}} \sqsubseteq \mathcal{A}^{\omega}$ is preserved, and

2. each vertex in $\mathcal{G}_{k+1}$ is at position that is strictly larger than ${\omega_k}^S$, which is at most ${\omega^S}$. □

Using a similar proof, we can show the correctness stated also for Algorithm 8. We recall our theorem and proceed to prove it below.

Theorem 8.1.3. *Let $\mathcal{G}$ be a (n, h)-parity game and let ϵ and ω be nodes that have even level and odd level h and $h+1$ in the two $\mathbb{N}$-labelled n-ary trees $\mathcal{T}^{\text{Even}}$ and $\mathcal{T}^{\text{Odd}}$ respectively. Procedure* McNZFast-Even$(\mathcal{G}, h, \epsilon, \omega)$*, initialised with the smallest*

$\mathcal{T}^{\text{Even}}$-decomposition of $\mathcal{G}$ for Steven and the smallest $\mathcal{T}^{\text{Odd}}$-decomposition of $\mathcal{G}$ for Audrey, terminates with the smallest Steven $\mathcal{T}^{\text{Even}}$-attractor decomposition for the Steven dominion and the smallest Audrey $\mathcal{T}^{\text{Odd}}$-attractor decomposition for the Audrey dominon in $\mathcal{G}$.

Proof. We claim that if the branching is not restricted, as is the case with Algorithm 8, we can add an additional invariant in our previous proof which states that every recursive call returns an attractor decomposition.

If the algorithm terminates on either the if or the else-if condition, we are done, since one player has a decomposition and for all other players the vertices are now at the side attractor of the other player.

If not, we need to argue that in each iteration of the for-loop of the algorithm the right subset of vertices is passed in the recursive subcall. Observe that each subtree of every complete n-ary tree also is a complete n-ary tree. We add this inductive invariant, with the additional condition that the trees are n-ary complete trees for both players. We further state that the necessary condition is the following:

For each subgame that is recursively passed, and for each tree that is corresponding to that level of recursion, the Audrey and Steven dominions in that subgame must have an attractor decomposition with respect to these trees.

Indeed the complete n-ary tree fits this criterion, as they are

- large enough to fit all attractor decompositions for any subgame and
- their subtrees passed are also complete n-ary trees.

Consider the decomposition $\mathcal{D}^{\epsilon}_{\text{Even}}$ and $\mathcal{D}^{\omega}_{\text{Odd}}$ smaller than $\mathcal{A}^{\epsilon}$ and $\mathcal{A}^{\omega}$. We will show that after the i^{th} iteration, the Audrey decomposition maintained is such that all vertices in $[\![\mathcal{D}^{\omega_i}_{\text{Odd}}]\!]$ form an attractor decomposition for Audrey and are the same as the attractor decomposition $\mathcal{A}^{\omega_i}_{\text{Odd}}$.

In the first step, we compute an attractor to vertices of priority h in the subgame $\mathcal{G}_i$. Observe that the dominion of Audrey $\left[\mathcal{A}^{\omega_i}\right]$ does not intersect with this attractor. This has been argued in the previous proof using Proposition 7.2.6 in Chapter 7. Moreover, since $\left[\mathcal{A}^{\omega_i}\right]$ is a trap for Steven, and hence all Audrey dominions in $\mathcal{G}_i$ are preserved in this trap.

This recursive subcall therefore identifies this set of vertices in $\left[\mathcal{A}^{\omega_i}\right]$ and hence also accurately computes S^{ω_i}, its side attractor and removes it from $\mathcal{G}_i$. Since the algorithm terminates only when we find an attractor decomposition, we also claim that both players on termination have an decomposition. □

Running time We show how our algorithm runs in time that is at most linear in the size of each of the tree, a significant reduction from other attractor-based algorithms with a quadratic dependence.

We show that procedure Univ-Even-Fast (resp. Univ-Odd-Fast) makes $\mathcal{O}\big(n^c \cdot \max\big(|\mathcal{T}^{\text{Odd}}|, |\mathcal{T}^{\text{Even}}|\big)\big)$ many recursive calls for a constant c. The time taken to perform each operation outside of the recursive calls is polynomial in n, thus making the running time of this algorithm $n^{\mathcal{O}(1)} \cdot \max\big(|\mathcal{T}^{\text{Odd}}|, |\mathcal{T}^{\text{Even}}|\big)$.

Lemma 8.2.2. *For an (n, h)-small parity game $\mathcal{G}$, the number of recursive calls made by procedure* Univ-Even-Fast *(resp.* Univ-Odd-Fast*) with trees $\mathcal{T}^{\text{Odd}}$ and $\mathcal{T}^{\text{Even}}$ is bounded by $\mathcal{O}\big(n^c \cdot \min\big(|\mathcal{T}^{\text{Odd}}|, |\mathcal{T}^{\text{Even}}|\big)\big)$ for a constant c. The time taken to perform each operations outside of the recursive subcalls is a polynomial in n.*

Proof. Since the operations performed themselves take only polynomial time, the running time of the algorithm is dominated, up to a polynomial factor, by the number of recursive calls made. Therefore, the total running time would be at most a product of a polynomial in n and h and the number of recursive calls made.

We call a recursive call *trivial* if it is an empty recursive call. If a subcall at level ϵ for Steven and ω for Audrey is empty, of course, we do not make any further recursive calls to its children, and we move to the following recursive call to next sibling in the tree for ϵ. This does not contribute to a large overhead because with the help of specific data structures that keep track of the next sibling whose parts in decomposition is non-empty, looking through all siblings could be avoided.

If a recursive subcall is non-trivial, then we see that there is a strict increase in the decomposition of either Audrey or Steven in this recursive call as we have shown in the proof of Lemma 8.2.1. This is due to the invariant that the decomposition is always modified in such a way that the intersection of $\big[\mathcal{D}^{\epsilon}\big]$ and $\big[\mathcal{D}^{\omega}\big]$ is empty. Since we begin with a non-empty set, and we only perform monotonic operations on the decomposition, we prove our desired bound on the running time.

We can further show an improved running time of $\mathcal{O}\big(n^c \cdot \min\big(\mathcal{T}^{\text{Odd}}, \mathcal{T}^{\text{Even}}\big)\big)$ with a small modification. Let us call a recursive call *accelerating* if this call does not enter the else statement of the algorithm. That is, the decomposition is already an attractor decomposition for at least one player.

Observe that if a recursive call is non-accelerating and nontrivial, then for *both* players, there is at least one vertex that is such that in any attractor decomposition larger than the current decomposition, this vertex is not in the current part. Identifying such vertices can be performed alongside the check to verify whether we already have an attractor decomposition. Once we identify such a vertex for both

the Audrey and Steven decomposition, we can increase it minimally at the end of each recursive call if its position remains unchanged.

This forces an increase for *both* the decompositions, ensuring that our algorithm terminates in time $\mathcal{O}\big(n^c \cdot \min\big(\mathcal{T}^{\mathrm{Odd}}, \mathcal{T}^{\mathrm{Even}}\big)\big)$. □

8.3 Outlook

A weakness of all quasi-polynomial symmetric attractor-based algorithms—including ours—is that they may output correct winning sets, but without constructing winning strategies. This is a major shortcoming in the context of synthesis, where winning strategies correspond to the desired controllers. We argue that our technique, which is based on computing decompositions that are under-approximations of the least attractor decompositions, allows one to tackle this weakness with a modest additional computational cost. If the algorithm terminates with a decomposition that is not an attractor decomposition, then the decomposition obtained can serve as a starting point to make further progress. We can then run the asymmetric algorithm or the strategy improvement algorithm from the previous chapters for each player. This would repeatedly modify the decomposition until an attractor decomposition is obtained. This addition does not increase the worst-case asymptotic running time by more than a polynomial factor.

We have illustrated that our technique, when applied to the standard McNaughton-Zielonka algorithm, yields an algorithm that can solve some hard examples [Fri11] in polynomial time. Other families of hard examples [vD19, BDM20] should also be analysed. Should our technique also solve them in polynomial time, delving into the structural challenges hindering the construction of hard examples could potentially offer fresh insights to tackle central questions in the algorithmic study of parity games.

Part III

Chapter 9

Rabin games and colourful trees

In this chapter, we shift our focus from parity games to Rabin games, a generalisation of parity games. Rabin games lie at the core of reactive synthesis for ω-regular or LTL specifications, and efficient algorithms for Rabin games are of practical interest in synthesis tools [Pnu77, BJP$^+$12].

Rabin automata already appear in McNaughton's solution of Church's synthesis problem [Chu57, McN66] and in Rabin's proof of the decidability of SnS [Rab69], where it was first defined in the setting of infinite trees. To solve Church's synthesis problem for ω-regular specifications, represented by non-deterministic Büchi automata, there are two (polynomial-time equivalent) approaches: either reduce to the emptiness problem for Rabin tree automata or solve a Rabin game.

As discussed earlier, Rabin conditions are also suitable specifications for *general fairness constraints* [FK84]. Klarlund and Kozen [KK91] defined Rabin measures over graphs and applied them to prove termination of a program under a general fairness constraint. Indeed, the acceptance condition that defines *strong fairness*, i.e. if a given set of actions (edges) is enabled infinitely often (the source vertex are seen infinitely often), it is taken infinitely often, is naturally expressed by the complement of the Rabin condition, called the Streett condition.

We briefly recall the algorithms to solve Rabin games in Table 9.1 discussed in the introduction of the thesis, which details the history of such algorithms.

Calude, Jain, Khoussainov, Li and Stephan's algorithm [CJK$^+$22] to solve parity games also gave fixed-parameter tractable (FPT) algorithms for Muller games on k colours, where the dependence on the number of colours is of the order k^{5k}. It is known that a Rabin game with k colours can be translated into a Muller game using at most $2k$ colours (but with the number of vertices increased by a factor of k) thereby giving a direct algorithm to solve Rabin and Streett games in time

Year	Algorithm	Time complexity
1988	Emerson and Jutla [EJ88, EJ99]	$\mathcal{O}(nk)^{3k}$
1989	Pnueli and Rosner [PR89]	$\mathcal{O}(nk)^{3k}$
2001	Kupferman and Vardi [KKV01]	$\mathcal{O}(mn^{2k} \cdot k!)$
2005	Horn [Hor05]	$\mathcal{O}(mn^{2k} \cdot k!)$
2006	Pieterman and Pnueli [PP06]	$\mathcal{O}(mn^{k+1}k \cdot k!)$
2017	Calude et al., [CJK+22]	$\mathcal{O}(nk^2 \cdot k!m)^{\log k+6}$
2017	Jurdziński and Lazic [JL17] or Fearnley et al. [FJdK+19]	$\mathcal{O}(nm \cdot k!^{2+o(1)})$

Table 9.1: Algorithms that solve Rabin games

proportional to $\mathcal{O}\Big((2^k)^{5\cdot 2^k}\Big)$. They further remarked that a more efficient way to solve Rabin game would be to convert a Rabin game to a parity game rather than a Muller game and solve the obtained parity game.

A Rabin game with n vertices, m edges, and k colours can be reduced to a parity game with $N = n\cdot k!$ vertices, $M = m\cdot k!$ edges, and $K = 2k+1$ colours [GH82] and to solve a Rabin game, one can instead solve this larger parity game (the priority on these games however appear on edges). While using the algorithm proposed by Calude et al. would exacerbate the space complexity of solving an already exponentially large parity game, choosing to use state-of-the-art parity game algorithms that improve on the space efficiency—such as the one by Jurdziński and Lazić [JL17]—enables the solution of Rabin games in time $\mathcal{O}\Big(\max\Big\{MN^{2.38}, 2^{\mathcal{O}(K\log K)}\Big\}\Big)$. However, it is worth noting that exponential space requirement remains a characteristic of such solutions.

On substitution N, M, and K, the algorithm of Jurdziński and Lazić would take time $\mathcal{O}\Big(mn^{2.38}(k!)^{3.38}\Big)$. However, observe that the parity game obtained from a Rabin game is such that the number of vertices $N = n \cdot k!$ is much larger than the number of colours $K = 2k + 1$. This results in $K \in o(log(N))$ as k increases. For cases where the number of vertices of the resulting parity game is much larger than the number of priorities, say the number of colours K is $o(\log(N))$, Jurdziński

and Lazić's algorithm solves these parity games obtained from Rabin games in time $\mathcal{O}\big(nmk!^{2+o(1)}\big)$ [JL17, Theorem 7]. Closely matching this are the running times in the work of Fearnley et al. [FJdK+19] who provide, among other bounds, a quasi-bilinear bound of $\mathcal{O}\big(MN\mathfrak{a}(N)^{\log\log N}\big)$, where $\mathfrak{a}$ is the inverse-Ackermann function. Therefore, the algorithm with the best worst-case time complexity has at least a $(k!)^{2+o(1)}$ dependence on the factorial of k in its running time and takes a space proportional to $(nk^2 \cdot k!)\log(nk^2 \cdot k!)$, which again has a $k!$ dependence.

There is a $\widetilde{\mathcal{O}}(nk)$ space algorithm by Piterman and Pnueli [PP06] which holds the crown for the smallest space requirements so far. However, this algorithm has a $\mathcal{O}(n^k \cdot k!)$ worst-case runtime (although the space complexity is claimed to be $\mathcal{O}(nk)$ in their paper, the exact bit complexity one needs would include an extra $\log n$ factor to encode each vertex and $\log k$ bits to encode each colour).

In this chapter, we provide an algorithm that breaks through the $2 + o(1)$ barrier, while simultaneously using polynomial space, $\mathcal{O}(nk \log n \log k)$, improving on Piterman and Pnueli's algorithm as well as algorithms that convert to a parity game, to give an FPT algorithm for Rabin games.

Our algorithm is achieved by firstly arguing that Rabin games have a colourful decomposition. Colourful decompositions are further extensions of attractor decompositions, modified to suit Rabin games. We then observe that these colourful decompositions naturally correspond to colourful trees. These colourful trees are a modified version of the pointer trees of Klarlund and Kozen. We then define *colourful universal trees*, which can embed any colourful tree. Just as Piterman and Pnueli's result generalised ranking techniques and progress measures for parity games, we generalise the notion of measures [KK91] and Jurdziński-Lazić universal trees [JL17] to obtain our algorithm.

9.1 Colourful trees and labelled colourful trees

To solve parity games, we had considered ordered trees, defined inductively as a tuple consisting of other ordered trees. To tackle Rabin games, we branch out to consider a closely related notion to ordered trees, which we call colourful ordered trees.

(c_0, C)-Colourful ordered Trees. Let C be a finite set of colours and let $c_0 \notin C$ be a distinct colour assigned only to the root of a tree. Informally, a (c_0, C)-colourful ordered tree is a tree whose root is assigned the unique colour c_0, and and every other node has a colour from C associated to it. In addition, we expect that for

all paths, every node along this path from the root to a leaf is assigned different colours. However, as an exception, we allow some leaves to remain uncolored, which we denoted by a "dummy colour" $\perp \notin C$. Observe that our requirement about nodes having different colours ensures that the height of the tree is bounded by at most $|C|$.

Formally, for a finite set C, we recursively define (c_0, C)-colourful trees

- if $C = \varnothing$, we say $\mathcal{T}$ is a $(c_0, \varnothing)$-colourful tree if either
  - $\mathcal{T} = (c_0, \langle\rangle)$, and is the tree with a single node coloured by c_0 or
  - $\mathcal{T} = (c_0, \langle(\perp, \langle\rangle), \ldots, (\perp, \langle\rangle)\rangle)$, and is a tree where all the children of the root have the dummy colour $\perp$.
- if $C \neq \varnothing$, we say $\mathcal{T}$ is (c_0, C)-colourful tree if it is either
  - a $(c_0, \varnothing)$-colourful tree rooted at c_0; or
  - $\mathcal{T} = (c_0, \langle\mathcal{T}_1, \ldots, \mathcal{T}_\ell\rangle)$, and for all $i \in \{1, \ldots, \ell\}$, either there is a colour $c_i \in C$ and $\mathcal{T}_i$ is a $(c_i, C \setminus \{c_i\})$-colourful ordered tree, or $\mathcal{T}_i = (\perp, \langle\rangle)$. Note that these c_i need not be different from each other.

For two trees whose root colour is the same, we define concatenation similar to ordered trees, as the tree obtained by a sequential composition of the two trees. We define the *concatenation* of a (c_0, C_1)-colourful tree $\mathcal{T}_1 = \left(c_0, \left\langle \mathcal{T}_1^1, \ldots, \mathcal{T}_1^m \right\rangle\right)$ and a (c_0, C_2)-colourful tree $\mathcal{T}_2 = \left(c_0, \left\langle \mathcal{T}_2^1, \ldots, \mathcal{T}_2^\ell \right\rangle\right)$ as the $(c_0, C_1 \cup C_2)$-colourful tree $\left(c_0, \left\langle \mathcal{T}_1^1, \ldots, \mathcal{T}_1^m, \mathcal{T}_2^1, \ldots, \mathcal{T}_2^\ell \right\rangle\right)$, written as $\mathcal{T}_1 \cdot \mathcal{T}_2$. For a root colour c_0, a number $\ell \in \mathbb{N}$, and a (c_0, C)-colourful ordered tree $(c_0, \langle\mathcal{T}\rangle)$, we denote $\left(c_0, \left\langle \mathcal{T}^\ell \right\rangle\right)$ to be the tree with ℓ many copies of $\mathcal{T}$, $(c_0, \langle\mathcal{T}, \mathcal{T}, \ldots, \mathcal{T}\rangle)$. When (c_0, C) is clear from context, we simply say "colourful tree."

Embedding colourful trees. Given a (c_0, C)-colourful trees $\mathcal{U}$ and $\mathcal{T}$, we say $\mathcal{U}$ embeds $\mathcal{T}$ if $\mathcal{T} = (c_0, \langle\rangle)$, or $\mathcal{T} = (c_0, \langle\mathcal{T}_1, \ldots, \mathcal{T}_\ell\rangle)$ and $\mathcal{U} = (c_0, \langle\mathcal{U}_1, \ldots, \mathcal{U}_m\rangle)$ for some ℓ, m, and there is an increasing sequence of indices $1 \leqslant i_1 < i_2 < \cdots < i_\ell \leqslant m$ such that $\mathcal{U}_{i_j}$ embeds $\mathcal{T}_j$ recursively.

Labelled colourful trees. In what follows, we shall additionally label colourful trees with labels from some linearly ordered set. This is similar to the definition of labelled ordered trees in the preliminaries, but we avoid the recursive definition, as it is more convenient to define such labelled colourful trees as prefix-closed sets of sequences, using the isomorphism between a (recursively defined) tree and its set of paths.

Let $\mathbb{L}$ be a set of labels with a linear ordering $<_{\mathbb{L}}$ over the set labels $\mathbb{L}$. We use $C^{\perp}$ to denote the set obtained by adding the a new $\perp$ to C, that is, $C^{\perp} = C \cup \{\perp\}$. Let $\mathbb{L} \times C^{\perp}$ be the Cartesian product of $\mathbb{L}$ and $C^{\perp}$. We avoid the tuple to denote an element of $\mathbb{L} \times C^{\perp}$, and instead write αc or $\alpha\perp$ for elements $(\alpha, c), (\alpha, \perp) \in \mathbb{L} \times C^{\perp}$.

An $\mathbb{L}$-labelled (c_0, C)-colourful tree is a finite prefix-closed set of sequences over $\mathbb{L} \times C^{\perp}$ such that only the maximal elements of this set contain, as a term, elements from $\mathbb{L} \times \{\perp\}$.

Given an element $\tau_0 \in \mathbb{L} \times C^{\perp}$ and a sequence $\langle \tau_1, \tau_2, \ldots, \tau_j \rangle$ of elements with each element from $\mathbb{L} \times C^{\perp}$, we use $\odot$ to denote concatenation to the tuple, where we say $\tau_0 \odot \langle \tau_1, \tau_2, \ldots \tau_j \rangle = \langle \tau_0, \tau_1, \tau_2, \ldots \tau_j \rangle$. We extend this notation to sets of sequences $\mathcal{L}$, by defining $\tau_0 \odot \mathcal{L}$ as the *prefix-closure* of the set of sequences $\{ \langle \tau_0, \tau_1, \tau_2, \ldots \tau_j \rangle \mid \langle \tau_1, \tau_2, \ldots \tau_j \rangle \in \mathcal{L} \}$.

We say that an $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$ is an $\mathbb{L}$-labelling of a (c_0, C)-colourful ordered tree $\mathcal{T}$ if

- if $\mathcal{T} = \left(c_0, \left\langle (\perp, \langle\rangle)^m \right\rangle\right)$, then $\mathcal{L}$ is the set $\{\langle\rangle\} \cup \{\langle \alpha_1 \perp \rangle, \ldots, \langle \alpha_m \perp \rangle\}$ for some $\alpha_1 <_{\mathbb{L}} \alpha_2 <_{\mathbb{L}} \cdots <_{\mathbb{L}} \alpha_m \in \mathbb{L}$, or
- if $\mathcal{T} = (c_0, \langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle)$, then $\mathcal{L}$ is the set

$$\bigcup_{i=1}^{m} \alpha_i c_i \odot \mathcal{L}_i$$

  for some increasing values $\alpha_1 \leqslant_{\mathbb{L}} \alpha_2 \leqslant_{\mathbb{L}} \ldots \leqslant_{\mathbb{L}} \alpha_m$ in $\mathbb{L}$, such that for all i in $\{1, \ldots, m\}$,

  - $\mathcal{T}_i$ is a $(c_i, C \setminus \{c_i\})$-colourful tree and $\mathcal{L}_i$ is an $\mathbb{L}$-labeling of $\mathcal{T}_i$,
  - $c_i \in C^{\perp}$, and
  - whenever the elements from the label set $\alpha_i = \alpha_{i+1}$, the corresponding colours $c_i \neq c_{i+1}$.

Note that the root colour c_0 of the colourful tree $\mathcal{T}$ does not appear in $\mathcal{L}$; instead of tracking c_0 along with $\mathcal{L}$ explicitly, we implicitly assume the root colour of the tree $\mathcal{L}$ above is c_0.

We refer to elements of a labelled colourful tree $\mathcal{L}$ as *nodes* of the tree. For two nodes n_1 and n_2 in $\mathcal{L}$, we define *the greatest common ancestor*, written $\mathsf{GCA}(n_1, n_2)$, as the longest common prefix of n_1 and n_2. We define n_1 as an *ancestor* of n_2 if $n_1 = \mathsf{GCA}(n_1, n_2)$. In particular, n_1 is a parent of n_2, written $n_1 = \text{parent}(n_2)$, if n_1 is the largest node other than n_2 such that $n_1 = \mathsf{GCA}(n_1, n_2)$; then we say that n_2 is a child of n_1.

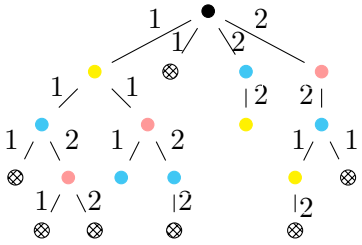


Figure 9.1: A colourful tree.

The *colouring* of a node is defined as the last colour that occurs in the sequence: For the empty sequence $\langle\rangle$, we define the colour of $\langle\rangle$, denoted by $\mathsf{colour}(\langle\rangle)$ as c_0 and $\mathsf{colour}\left(\left\langle \alpha_1 c_{i_1}, \ldots, \alpha_j c_{i_j} \right\rangle\right) = c_{i_j}$. Furthermore, we define $\mathsf{ColourSet}$ as a function from the set of nodes of a labelled colourful tree to the subset of colours $\mathcal{P}(C \cup \{c_0, \bot\})$. This function maps a node to the set of colours seen from the root to that node $\mathsf{ColourSet}(n) = \left\{\mathsf{colour}(n') \mid n' = \mathsf{GCA}(n', n)\right\}$. For example, consider the colourful tree in Fig. 9.1 and the node $\langle 1\bullet, 1\bullet, 2\bullet, 2\otimes\rangle$ in it. The colour of the node, denoted by $\mathsf{colour}(\langle 1\bullet, 1\bullet, 2\bullet, 2\otimes\rangle) = \otimes$ and $\mathsf{ColourSet}(\langle 1\bullet, 1\bullet, 2\bullet, 2\otimes\rangle) = \{\bullet, \bullet, \bullet, \bullet, \otimes\}$. Whereas for the node $\langle 1\bullet, 1\bullet, 1\bullet\rangle$, we have $\mathsf{colour}(\langle 1\bullet, 1\bullet, 1\bullet\rangle) = \bullet$ and $\mathsf{ColourSet}(\langle 1\bullet, 1\bullet, 1\bullet\rangle) = \{\bullet, \bullet, \bullet, \bullet\}$.

Ordering of nodes. We define a total linear order $\prec_{\mathcal{L}}$ on the nodes of a fixed colourful tree $\mathcal{L}$. First, we fix some arbitrary linear order on the set $C^{\bot}$ such that $\bot$ is the largest element in the ordering. We compare elements by extending the linear order $<_{\mathbb{L}}$ over $\mathbb{L}$ and an arbitrary fixed order $<$ over C to a linear order over the set $\mathbb{L} \times C^{\bot}$ lexicographically. For two elements $\alpha_1 c_1$, $\alpha_2 c_2$ in $\mathbb{L} \times C^{\bot}$, we define $\alpha_1 c_1 < \alpha_2 c_2$ if either $\alpha_1 <_{\mathbb{L}} \alpha_2$ or $\alpha_1 = \alpha_2$ and $c_1 < c_2$.

For two nodes $n_1, n_2 \in \mathcal{L}$, we define $n_2 \prec_{\mathcal{L}} n_1$ if either n_2 is a strict prefix of n_1, or if n_1 is lexicographically larger than n_2 when viewed as sequences over $\mathbb{L} \times (C \cup \{\bot\})$. We remove the subscript $\mathcal{L}$ when the labelled colourful tree is clear from the context.

Pictorially, the defined ordering on a tree decreases when we go from a child to a parent, or we go "left" in the tree, but otherwise increases. In Figure 9.1, for example here is the order with respect to $\prec$ on the following nodes $\langle 1\bullet\rangle \prec \langle 1\bullet, 1\bullet\rangle \prec \langle 1\otimes\rangle \prec \langle 2\bullet, 2\bullet\rangle$.

Example 6. *In Fig. 9.1, we show a* $(\bullet, \{\bullet, \bullet, \bullet\})$*-colourful tree, where* $\otimes$ *represents the dummy colour. For a fixed ordering on the set of colours* $\bullet < \bullet < \bullet < \otimes$*, a labelling of this tree over* $\mathbb{L} = \{1, 2\} \subseteq \mathbb{N}$ *is the prefix closure of the set* $\{\langle 1\bullet, 1\bullet, 1\otimes\rangle,$

$\langle 1\bullet, 1\bullet, 2\bullet, 1\otimes\rangle, \langle 1\bullet, 1\bullet, 2\bullet, 2\otimes\rangle, \langle 1\bullet, 1\bullet, 1\bullet\rangle, \langle 1\bullet, 1\bullet, 2\bullet, 2\otimes\rangle, \langle 1\otimes\rangle, \langle 2\bullet, 2\bullet\rangle, \langle 2\bullet, 1\bullet, 1\bullet, 1\otimes\rangle, \langle 2\bullet, 1\bullet, 2\otimes\rangle\}$.

9.2 Shape of a Rabin game

In this section, our aim is to understand the Rabin acceptance condition on games. We define such acceptance conditions and provide a witness that we call *a colourful decomposition* of a game in which Steven wins from all vertices. We further remark that we can equivalently obtain a local witness called a *Rabin measure* for such games.

Rabin game. Recall that a (c_0, C)-colourful *Rabin game* $\mathcal{G}$ consists of

1. an *arena* of a (sink-free) directed graph (V, E),
2. a *start vertex* $v_0 \in V$ and
3. a partition of V into V_S and V_A, the vertices of two players, Steven and Audrey, respectively.
4. a finite set C of *colours*, and a special colour $c_0 \notin C$, and
5. for each vertex $v \in V$, a set of *good* colours $G_v \subseteq C \cup \{ c_0 \}$ for v and a set $B_v \subseteq C$ of *bad* colours for v.

Observe that $c_0 \notin B_v$ for any v. We call a colour c in G_v a *good* colour for v, and a colour in B_v a *bad* colour for v.

An infinite path in a Rabin game $\mathcal{G}$ satisfies the *Rabin condition* if there is some colour c in $C \cup \{c_0\}$ such that c is a good colour for some v seen infinitely often along the path and c is not a bad colour for any v seen infinitely often along the path.

Defined similar to parity games, a *positional strategy* σ for Steven for a Rabin game $\mathcal{G}$ is a subset of edges outgoing from Steven's set of vertices V_S along with all of Audrey's edges in the game. We denote the graph restricted to a strategy σ for Steven by $\mathcal{G}|_\sigma$ and it is defined as the Rabin graph over the same vertex set with a new edge relation that consists exactly of the edges in σ.

We define the following parameters: n is the number of vertices, m is the number of edges, and $k = |C \cup \{c_0\}|$ is the number of colours (also called the *index*).

Remark 6. *Instead of Rabin condition usually described by pairs of subsets of vertices associated to a colour, we talk about sets of colours associated to a vertex instead. This is an equivalent representation in terms of size.*

Colourful decomposition. We first define another extension of attractor decompositions, suitable for Rabin games, called *colourful decompositions*. These colourful decompositions of a Rabin graph highlight a recursive structure that captures the acceptance of all plays of Steven in a way that relates naturally to colourful trees. Colourful decompositions generalise attractor decompositions defined in the preliminaries Chapter 2 for parity games to Rabin games.

Consider a (c_0, C)-colourful Rabin game $\mathcal{G}$ whose arena is the graph (V, E), and whose good and bad colours are denoted by G_v and B_v respectively. A (c_0, C)-colourful decomposition $\mathcal{D}$ of $\mathcal{G}$ is a recursive sub-division of vertices V into subsets of vertices defined as follows. If $C = \varnothing$, then we say $\mathcal{D} = \langle V \rangle$ is a (c_0, C)-colourful decomposition if V is the Steven attractor to the set containing all vertices v such that $c_0 \in G_v$. Else, if $C \neq \varnothing$, then we say

$$\mathcal{D} = \left\langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, \left(c_j, V_j, \mathcal{D}_j, A_j\right)\right\rangle$$

satisfies the following conditions:

1. A is the Steven attractor in the game $\mathcal{G}$ to the set of vertices v such that $c_0 \in G_v$;

and setting $\mathcal{G}_1 = V \setminus A$. For $i \in \{1, \ldots, j\}$, we have

2. V_i is a trap for Audrey in $\mathcal{G}_i$ in which the colour $c_i \notin B_v$ for all $v \in V_i$;
3. $\mathcal{D}_i$ is a $(c_i, C \setminus \{c_i\})$-colourful decomposition of the subgame $\mathcal{G} \cap V_i$;
4. A_i is the Steven attractor to V_i in $\mathcal{G}_i$;
5. $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$;

and we have $\mathcal{G}_{j+1} = \varnothing$.

We encourage the reader to refer to the corresponding definition of an attractor decomposition of a parity game given in the preliminaries. The definitions of attractor and trap are as defined for parity games in the preliminaries of the thesis.

Rabin measure. The Rabin measure, as with other progress measures, is based exclusively on local properties. This renders it perfect for creating algorithms to

solve these games. Indeed, we have a progress measure when each edge satisfies certain conditions. We fix a (c_0, C)-colourful Rabin game $\mathcal{G}$ with the underlying graph (V, E) with good colours for a vertex v denoted by G_v and the bad colours B_v. Let $\mathbb{L}$ be a linearly ordered set of labels, and let $\mathcal{L}$ be a $\mathbb{L}$-labelled (c_0, C)-colourful tree. We define $\mathcal{L}^\top = \mathcal{L} \cup \{\top\}$ by adjoining an element $\top$ to $\mathcal{L}$ and we extend the ordering $\prec$ to $\mathcal{L}^\top$, by declaring $\top$ to be greater than all nodes in the labelled colourful tree $\mathcal{L}$.

Consider a map $\mu : V \to \mathcal{L}^\top$. We call an edge $u \to v$ *consistent* with respect to μ, if either $\mu(u)$ is assigned to $\top$ or it satisfies conditions ($\mathsf{G}_\succ$ OR $\mathsf{G}_\downarrow$) AND B defined below.

- if $\mathsf{G}_\succ$, then $\mu(v) \prec \mu(u)$;
- if $\mathsf{G}_\downarrow$, then $\mathsf{GCA}(\mu(u), \mu(v)) = \mu(u)$ and $\mathsf{colour}(\mu(u)) \in G_u$;
- if B, then $\mathsf{ColourSet}(\mu(u)) \cap B_u = \varnothing$

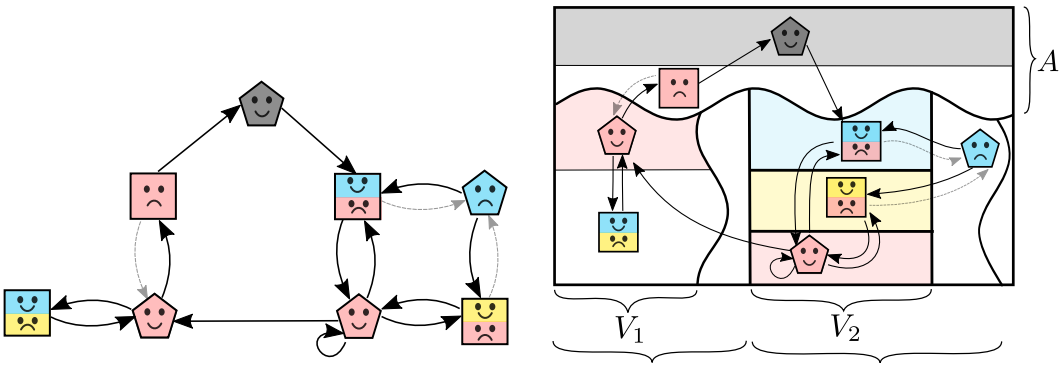
In words, $\mathsf{G}_\succ$ conveys that the measure μ decreases along the edge $u \to v$ and $\mathsf{G}_\downarrow$ says that the measure can increase to a descendent node but only when the colour of the node that is currently mapped to is a good colour for u. B says that none of the colours mapped to any ancestor of u is a bad colour for it.

If the map μ is clear from the context, we call an edge or a vertex consistent without mentioning the mapping. We say that the relation and function $\mathsf{GCA}(\cdot, \top)$ and $\mathsf{colour}(\top)$ are undefined, and the condition $\mathsf{G}_\downarrow$ or B are not satisfied when $\mu(v)$ is assigned to $\top$ and $\mu(u)$ is not assigned to $\top$.

We say that the map μ is a (c_0, C)-colourful *Rabin measure* for a game $\mathcal{G}$ if for Audrey's vertices, all edges outgoing from it are consistent, and there is at least one edge from each Steven vertex that is consistent with respect to μ.

Progress measures are values assigned to a state that denote how close this state is to satisfying a specific property [Kla90, KK91, Var96]. We provide a progress measure for Rabin games, from the work of Klarlund and Kozen [KK91], where they also provide a similar measure for Rabin graphs (equivalent to Rabin games when Audrey owns all vertices). Our definition of a Rabin measure combines the ideas of Klarlund and Kozen [KK91] as well as Jurdzinski and Lazic [JL17], following recent approaches to faster algorithms for parity games discussed extensively in Parts I and II of this thesis.

The crux of this section is Theorem 9.2.1 below which shows the equivalence between a Rabin measure, a colourful decomposition, and a Rabin game where Steven can win from all vertices.



(a) A $(\bullet, \{\bullet, \bullet, \bullet\})$-colourful Rabin game $\mathcal{G}$

(b) A $(\bullet, \{\bullet, \bullet, \bullet\})$-colourful Decomposition of $\mathcal{G}$

Figure 9.2: A Rabin game and its decomposition

Theorem 9.2.1. *The following three statements are equivalent.*

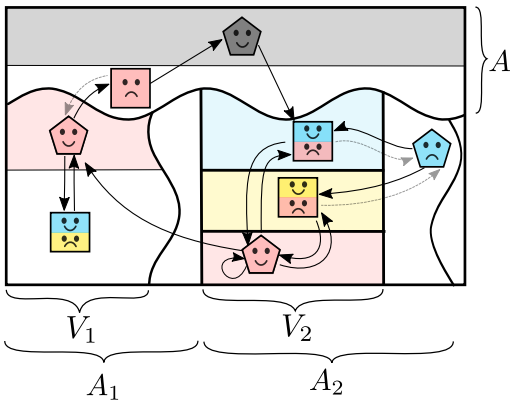
1. *Steven wins from all vertices in a (c_0, C)-colourful Rabin graph $\mathcal{G}$.*
2. *There is a (c_0, C)-colourful decomposition $\mathcal{D}$ of the vertices of $\mathcal{G}$.*
3. *There is an $\mathbb{L}$-labelled (c_0, C)-colourful Rabin measure for $\mathcal{G}$, where no vertex is mapped to $\top$ for some linearly ordered infinite set $\mathbb{L}$.*

The theorem above is proved by showing $(1 \implies 2)$ in Lemma 9.2.2, $(2 \implies 3)$ in Lemma 9.2.3 and finally $(3 \implies 1)$ in Lemma 9.2.6.

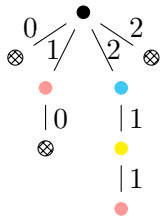
Example 7. *Consider the $(\bullet, \{\bullet, \bullet, \bullet\})$-colourful Rabin game in Fig. 9.2(a) and its $(\bullet, \{\bullet, \bullet, \bullet\})$-colourful decomposition in Fig. 9.2(b). The colours that are in the good set of each vertex are represented with a smiley face and those that are bad colours appear with a sad face. So, the leftmost vertex in the game $\mathcal{G}$ as in Fig. 9.2(a), has the singleton set $\{\bullet\}$ as the set of good colours and the set $\{\bullet\}$ as the set of bad colours, whereas the topmost vertex in Fig. 9.2(a) has $\{\bullet\}$ as the set of good colours and an empty set of bad colours. Steven wins fom all vertices in this game, and a positional strategy is highlighted in the same figure where the edges not in the strategy are dashed and grey. The decomposition of $\mathcal{G}$ in Fig. 9.2(b) can be represented as*

$$\mathcal{D} = \langle A, (\bullet, V_1, \mathcal{D}_1, A_1), (\bullet, V_2, \mathcal{D}_2, A_2) \rangle,$$

where the corresponding sets of vertices are represented in the picture, $\mathcal{D}_1$ is a $(\bullet, \varnothing)$-decomposition of the subgame induced by V_1, and $\mathcal{D}_2$ is a $(\bullet, \{\bullet, \bullet\})$decomposition by a subgame induced by V_2.



(a) A $(\bullet, \{\bullet, \bullet, \bullet\})$-Colourful decomposition of a Rabin game $\mathcal{G}$



(b) An $\mathbb{N}$-labelled $(\bullet, \{\bullet, \bullet, \bullet\})$-colourful tree into which the game has a Rabin measure.

Figure 9.3: A colourful decomposition and tree for Rabin measure

For the same game, we describe a Rabin measure μ obtained from the decomposition into the tree in Fig. 9.3(b), where the decomposition is reproduced in Fig. 9.3(a) next to the game for ease of reference. The only vertex with colour $\bullet$ as a good colour is assigned by μ to the root $\langle\rangle$. The other vertex in A is assigned by μ to the first child of the root node in the tree $\langle 0\otimes\rangle$. All vertices in V_1 are assigned by μ to one of the two nodes $\langle 1\bullet\rangle$ or $\langle 1\bullet, 0\otimes\rangle$. The vertices in V_2 are assigned to the subtree rooted at colour $\bullet$, to one of the nodes $\langle 2\bullet\rangle$, or $\langle 2\bullet, 1\bullet\rangle$ or $\langle 2\bullet, 1\bullet, 1\bullet\rangle$ depending on the colour which is a good colour of the vertex. Finally, the vertex in $A_2 \setminus V_2$ is assigned by μ to the last child of the root $\langle 2\otimes\rangle$.

Lemma 9.2.2. *Let $\mathcal{G}$ be (c_0, C)-colourful Rabin game where Steven wins from all vertices, then there is a (c_0, C)-colourful decomposition of $\mathcal{G}$.*

Proof. We construct such a decomposition, by inducting on $|C|$ and the number of vertices in $\mathcal{G}$.

Base case. If $C = \varnothing$, then the set of bad colours for each vertex is empty, that is, for all vertices v, the set $B_v = \varnothing$. Observe also that the attractor to the set B consisting of vertices v in the game such that $G_v = \{c_0\}$, is the entire game V. This is because the complement of a Steven Attractor is a trap for him. If therefore there are any vertices other than the attractor to B, Audrey can ensure that the play stays there and never visits a vertex v such that $G_v = \{c_0\}$. Therefore, the $(c_0, \varnothing)$-colourful decomposition is just $\langle V\rangle$.

Induction hypothesis. Let $\mathcal{G}$ be a (c_0, C)-colourful Rabin games $\mathcal{G}$ where Steven wins from everywhere and either $|C| < k$ or if there are strictly fewer than n vertices. Then $\mathcal{G}$ has a (c_0, C)-colourful decomposition

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

where $c_0 \notin B_v$ for all $v \in V$, and for all $v \in V$, if $c_0 \in G_v$ then $v \in A$.

Induction step. Suppose that $|C| = k$, there are n vertices, and the induction hypothesis holds. Consider all vertices $B = \{v \mid c_0 \in G_v\}$, and let A be the Steven attractor to the set B of vertices. The subgame $\mathcal{G}_1$ induced by $V \setminus A$ is a trap for Steven, and therefore he must win from any vertex in the subgame restricted to $\mathcal{G}_1$. Moreover, observe that there are no vertices v such that $c_0 \in G_v$ or $c_0 \in B_v$ for $v \in \mathcal{G}_1$.

Fix a Steven strategy that is winning for him in $\mathcal{G}_1$. Consider an SCC decomposition of the graph induced by the vertices of $\mathcal{G}_1$ using only the strategy edges for Steven, and all edges of Audrey. Consider a bottom SCC (an SCC from which there is no path to other SCCs) V_1 of the graph induced by $V \setminus A$. Consider a path π such that the set of all vertices visited by π infinitely often is exactly V_1. This path satisfies the Rabin condition, which implies that there is some colour c_1 such that $c_1 \notin B_v$ *for all* $v \in V_1$ and $c_1 \in G_v$ for some $v \in V_1$.

Therefore, by induction, there is a $(c_1, C \setminus \{c_1\})$-colourful decomposition of V_1, say $\mathcal{D}_1$. Let A_1 denote the Steven attractor to V_1 in the subgame $\mathcal{G}_1$.

Now consider the game $\mathcal{G}_2 = \mathcal{G}_1 \setminus A_1$, which has fewer vertices. We know again that $\mathcal{G}_2$ is a trap for Steven in $\mathcal{G}$. Moreover, there are no vertices v such that $c_0 \in G_v$ or $c_0 \in B_v$ for $v \in \mathcal{G}_2$, there must be a (c_0, C)-colourful decomposition.

Let this decomposition be:

$$\mathcal{D}' = \langle \varnothing, (c_2, V_2, \mathcal{D}_2, A_2), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle .$$

The first set of vertices is $\varnothing$ by induction hypothesis since there are no vertices v where c_0 is a good colour or a bad colour of v. We claim that

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), (c_2, V_2, \mathcal{D}_2, A_2), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

thus constructed from the sets defined above is a (c_0, C)-colourful decomposition. It is routine to verify that it satisfies all the properties of a decomposition by construction. □

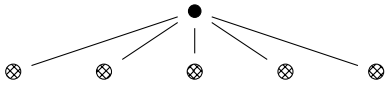


Figure 9.4: If $C \neq \varnothing$ and $n = 5$

Lemma 9.2.3. *Given a (c_0, C) colourful Rabin graph $\mathcal{G}$ on which we have a (c_0, C)-colourful decomposition $\mathcal{D}$, there is a $\mathbb{L}$-labelled (c_0, C)-colourful tree with Rabin measure for $\mathcal{G}$, where no vertex is mapped to $\top$.*

Proof of Lemma 9.2.3. The following is proved by induction on the size of C. Given a decomposition, we inductively obtain a tree and a corresponding mapping into the tree. We modify both the tree and Rabin measure thus obtained from the recursively defined decompositions and then merge them together. We later prove that indeed such a mapping defined is a Rabin measure.

Before we proceed, we define the Steven attractor length. For each vertex u, we say the Steven attractor length to a set B is t if t is the smallest number of steps such that Steven can ensure within t steps he can visit a vertex in B. Observe that all vertices in B have have attractor length 0, and the attractor length of any vertex in the Steven attractor to B is finite and at most $n - 1$.

Suppose $C = \varnothing$. The $(c_0, \varnothing)$-colourful decomposition $\mathcal{D} = \langle V \rangle$. The set of vertices V is exactly the Steven attractor to the set B consisting of all vertices v such that $c_0 \in G_v$. We consider an $\mathbb{L}$-labelled (c_0, C)-colourful tree obtained from $\mathcal{L}$, a tree with at most $n - 1$ nodes, all coloured $\bot$ other than the root c_0. More formally, it is the prefix closure $\mathcal{L}$ of the set of leaves $\{\langle \alpha_1 \bot \rangle, \ldots, \langle \alpha_t \bot \rangle\}$, where $\alpha_1 < \alpha_2 < \cdots < \alpha_t$, each α_i is an element of $\mathbb{N}$. A picture of this tree is in Fig. 9.4

All vertices $v \in B$ are mapped to the empty sequence denoted by $\langle\rangle$. For all vertices $v \notin B$, we define μ to be the i^{th} child of the root $\langle \alpha_i \bot \rangle$, where i is the Steven attractor length of v to B. Such a mapping μ has $B_v = \varnothing$ for all v, all the edges satisfies B trivially.

Now we consider all edges that would be used by Steven in the attractor strategy, along with all of Audrey's edges. We now show that such an edge $u \to v$ satisfies $\mathsf{G}_{\succ}$ or $\mathsf{G}_{\downarrow}$. Notice that if $c_0 \in G_u$, i.e, $u \in B$, then edge $u \to v$ satisfies $\mathsf{G}_{\downarrow}$. Else, $\mu(u) \succ \mu(v)$, since it must be the case that the Steven attractor distance to B from u is at least one more than this distance from v (this is an edge in the strategy graph). Therefore if $\mu(u) = \langle \alpha_j \bot \rangle$ and $\mu(v) = \langle \alpha_i \bot \rangle$, then $\alpha_j > \alpha_i$ and therefore $\mu(u) \succ \mu(v)$.

Suppose $C \neq \varnothing$. We have a (c_0, C)-colourful decomposition $\mathcal{D}$, where

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle .$$

Then for each V_i, since it has a $(c_i, C \setminus \{c_i\})$-colourful decomposition, by induction, we have a mapping μ_i to an $\mathbb{L}$-labelled $(c_i, C \setminus \{c_i\})$-colourful tree $\mathcal{L}_i$.

We give a Rabin measure μ into the labelled colouuful tree $\mathcal{T}$ below

$$\left\{ \left\langle \alpha_1^0 \bot \right\rangle, \ldots, \left\langle \alpha_t^0 \bot \right\rangle \right\} \bigcup_{i=1}^{j} \left\{ \alpha_0^i c_i \odot \mathcal{L}_i, \left\langle \alpha_1^i \bot \right\rangle, \ldots, \left\langle \alpha_t^i \bot \right\rangle \right\}$$

where α_ℓ^i are elements from $\mathbb{L}$ such that $\alpha_\ell^{i_1} < \alpha_{\ell'}^{i_2}$ if $i_1 < i_2$ and $\alpha_{\ell_1}^i < \alpha_{\ell_2}^i$ if $\ell_1 < \ell_2$. The tree is such that we add in order, $t = n - 1$ many children coloured with $\bot$ to the root followed by the recursively obtained (c_i, C_i)-colourful tree $\mathcal{L}_i$ to the root for each i. A picture of the tree is given in Fig. 9.5 where the pink and the blue triangles represent the trees obtained recursively. We define $\mu(u)$ from a decomposition $\mathcal{D}$ above as follows.

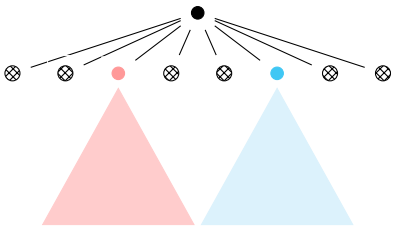


Figure 9.5: Suppose $C \neq \varnothing$

- If $u \in A$ and $c_0 \in G_u$, then $\mu(u) = \langle\rangle$.
- If $u \in A \setminus \{v \in A \mid c_0 \in G_v\}$, we define $\mu(u) = \left\langle \alpha_\ell^0 \bot \right\rangle$ where ℓ is the attractor length from u to the set consisting of all vertices v such that $c_0 \in G_v$.
- For vertices $u \in V_i$, we define $\mu(u) = \alpha_0^i c_i \odot \mu_i(u)$.
- For vertices $u \in A_i \setminus V_i$, we define $\mu(u) = \left\langle \alpha_\ell^i \bot \right\rangle$ where ℓ is the Steven attractor length from u to V_i in $\mathcal{G}_i$.

We show that the μ defined above satisfies the conditions required for it to be a Rabin measure by instead showing that each edge in the graph $\mathcal{G}$ is consistent. First, we make the following observation about the defined mapping. For the rest of the proof, we sometimes write A_0 to also mean A and let V_0 represent the set

$\{v \in A \mid c_0 \in G_v\}$. Let $\mathcal{G}_i$ be defined similarly to the definition of a decomposition, where $\mathcal{G}_1 = V \setminus A$, and $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$. Furthermore, we define $\mathcal{G}_0$ as the game $\mathcal{G}$.

> $(*)$ For $i \in \{0, 1, \ldots, j\}$, any vertex in $u \in V \setminus \mathcal{G}_i$ is such that $\mu(u) \prec \mu(v)$ for all $v \in \mathcal{G}_i$.

Moreover, we fix a Steven attractor strategy in each A_i, restrict Steven to only this strategy edges from his vertices and argue that all such edges are consistent, to argue that μ is consistent.

- If $u \in A$ and $c_0 \in G_u$, since $\mu(u) = \langle\rangle$, for such a vertex any edge $u \to v$ satisfies $\mathsf{G}_{\downarrow}$, since the root is coloured with c_0, and also satisfies B since $c_0 \notin B_u$.
- If $u \in A$ and $c_0 \notin G_u$, then $\mu(u) = \left\langle \alpha^0_\ell \bot \right\rangle$, where ℓ is the length such that u is in the ℓ is the Steven attractor length. Therefore, neighbours in the game restricted to the Steven attractor strategy, must be assigned to $\alpha^0_{\ell_1}$ where $\ell_1 < \ell$.
- If $u \in A_i \setminus V_i$, for $i \in \{0, 1, \ldots, j\}$, and suppose $\mu(u) = \left\langle \alpha^i_\ell \bot \right\rangle$ we show that edges from u satisfies $\mathsf{G}_{\succ}$.
  - If $v \in V \setminus \mathcal{G}_i$, then edge $u \to v$ satisfies $\mathsf{G}_{\succ}$ from $(*)$, as all vertices in $V \setminus \mathcal{G}_i$ are mapped to a node strictly smaller than $\mu(u)$ already.
  - If $v \in A_i$, all paths using the Steven attractor strategy A_i in $\mathcal{G}_i$ (as defined in the definition of a decomposition) leads to a vertex in V_i.
  - If $v \in V_i$ then by definition it is mapped to a descendent of α^i_0 and is therefore assigned to a value smaller than $\left\langle \alpha^i_\ell \bot \right\rangle$, and satisfies $\mathsf{G}_{\succ}$. If not, then v is a neighbour of u in $A_i \setminus V_i$ and must have its Steven attractor distance (to V_i in $\mathcal{G}_i$) to be strictly smaller than that from u.

  Therefore for any neighbour v, from our assignment of μ, it must be the case that $\mu(v) = \left\langle \alpha^i_{\ell_1} \bot \right\rangle$, where $\ell > \ell_1$, and hence $\mu(u) \succ \mu(v)$. Observe that all edges from u also satisfies B because the only ancestor of $\mu(u)$ is $\langle\rangle$, and is coloured with c_0 and $c_0 \notin B_v$ for any v, and therefore specifically $c_0 \notin B_u$.
- If $u \in V_i$ for $i \in \{1, \ldots, j\}$, for all edges $u \to v$, v is either in V_i or in $V \setminus \mathcal{G}_i$ since there are no paths using the Steven strategy fixed from V_i to $\mathcal{G}_i \setminus V_i$ (V_i is a trap for Audrey in $\mathcal{G}_i$). If $v \in V \setminus \mathcal{G}_i$, we know $\mu(u) \succ \mu(v)$ from $(*)$, and thus $\mathsf{G}_{\succ}$ is satisfied. On the other hand if $v \in V_i$, then $\mu_i(u)$ and $\mu_i(v)$ are both defined. If edge $u \to v$ satisfies $\mathsf{G}_{\downarrow}$ with respect to μ_i, then it continues to be satisfied in μ since $\mathsf{colour}(\mu(u)) = \mathsf{colour}(\mu_i(u))$. Otherwise, the edge

$u \to v$ satisfies $\mathsf{G}_{\succ}$ in μ_i, i.e. $\mu_i(u) \succ \mu_i(v)$. Then $\mu(u) \succ \mu(v)$ since μ appends the same value to the beginning of $\mu_i(u)$ and $\mu_i(v)$. Thus, $\mathsf{G}_{\succ}$ is satisfied with respect to μ.

Observe that $\mathsf{ColourSet}(\mu(u)) = \mathsf{ColourSet}(\mu_i(u)) \cup \{c_i\}$. Also, notice that from the definition of a decomposition $c_i \notin B_u$ for any $u \in V_i$. So, if $\mathsf{ColourSet}(\mu_i(u)) \cap B_u = \varnothing$, then $B_u \cap \mathsf{ColourSet}(\mu(u)) = \varnothing$. Thus, B is also satisfied by edge $u \to v$. □

For the proof of Lemma 9.2.6 which would show how a Rabin measure serves as a witness that all infinite paths in a Rabin graph satisfy the Rabin condition, we require the following two simple facts on trees. These hold in general for all ordered trees and not just colourful ordered trees. First one in Proposition 9.2.4 says that among two nodes in a tree, any ancestor of the larger node is always either an ancestor of a smaller node or is also larger than the smaller node. The latter proposition is about an infinite sequence of nodes in a tree where two consecutive nodes satisfy some given properties.

Proposition 9.2.4. *Any ancestor t of t' is such that for any other node $t'' \prec t'$, either t is an ancestor of t'' or t is strictly larger than t''.*

Proposition 9.2.5 (Lemma 1, [KK91])**.** *Consider an infinite sequence ρ of nodes from $\mathcal{L}$, an $\mathbb{L}$-labelled (c_0, C)-colourful tree, where $\rho = t_0, t_1, \ldots, t_i, \ldots$. Suppose for all $j \in \mathbb{N}$, if*

(i) either $t_j \succ t_{j+1}$ or

(ii) t_j is an ancestor of t_{j+1}

then the smallest element of the sequence, denoted by t_{inf} must be

1. *the largest common ancestor of t_i and t_{i+1} infinitely often*
2. *an ancestor of all but finitely many t_is.*

Proof. Let p be the position after which all t_k such that $k > p$ are such that $t_k \in \mathsf{Inf}(\rho)$. Without loss of generality, assume $t_p = t_{\min}$. Clearly, $t_{p+1} \succeq t_p$, since it is the smallest among $\inf\{\rho\}$.

1. We recall that $t_p \succ t_{p+1}$ or t_p is an ancestor of t_{p+1}. And we can conclude that t_p is an ancestor of t_{p+1}. Since after position p, each element occurs infinitely many times, we have that $t_{\min}$ is the largest common ancestor of $t_{\min}$ and all its occurrences t_i and its successors t_{i+1}.

2. We also argue that $t_{\min}$ is an ancestor of all t_j for $j \geqslant p$. Let the next occurrence of $t_{\min}$ in ρ be at t_q, where $q > p$. We will show that for all $p \leqslant j \leqslant q$, $t_{\min}$ is an ancestor of t_j, or equivalently that t_p is an ancestor of t_j. Indeed, consider $t_p, t_{p+1}, \ldots, t_q$.

   We show $t_p = \mathsf{GCA}(t_p, t_p) = \mathsf{GCA}(t_p, t_{p+1}) = \cdots = \mathsf{GCA}(t_p, t_q) = t_p$. We proceed by induction. In the base case, trivially t_p is an ancestor of t_p. We assume as the induction hypothesis that t_p is an ancestor of t_{p+i}. We know that t_{p+i} and t_{p+i+1} satisfy either $t_{p+i} \succ t_{p+i+1}$ or t_{p+i} is an ancestor of t_{p+i+1}.

   In the latter case t_{p+i} is an ancestor of t_{p+i+1}. By the induction hypothesis we have that t_p is an ancestor of t_{p+i}. Therefore, we conclude that t_p is an ancestor of t_{p+i+1}.

   In the former case, we invoke Proposition 9.2.4 with $t := t_p$, $t' := t_{p+i}$ and $t'' := t_{p+i+1}$. We consequently get either $t_{p+i+1} \prec t_p$ or t_p is an ancestor of t_{p+i+1}. Since $t_p = t_{\min}$, this gives us t_p is an ancestor of t_{p+i+1} concluding our claim. □

Lemma 9.2.6. *If there is an $\mathbb{L}$-labelled (c_0, C)-colourful Rabin measure for a (c_0, C)-colourful Rabin graph $\mathcal{G}$ and no vertex is mapped to $\top$, then the game is winning from all vertices.*

Proof. Let the Steven strategy consist exactly of the edges that are consistent with respect to μ from V to the (c_0, C)-colourful tree $\mathcal{L}$. Consider an infinite path $\pi = v_0 \to v_1 \to \cdots \to v_j \to \ldots$ in $\mathcal{G}$. We define the infinite sequence $\mu(\pi) = \mu(v_0), \mu(v_1), \ldots, \mu(v_j), \ldots$, obtained by taking the image of the run by μ on the colourful tree. In this colourful tree, consider t to be the smallest element among the elements of $\mathcal{L}$ that occur infinitely often in the sequence $\mu(\pi)$, and let $c = \mathsf{colour}(t)$. For such a t, we show

1. t is not coloured with $\bot$;
2. $c \in G_v$, for infinitely many v in π, and
3. $c \notin B_v$ for each v occurring after some finite prefix in π,

to conclude that π is satisfies the Rabin condition. Before we begin the rest we first remark that from conditions $\mathsf{G}_{\succ}$ or $\mathsf{G}_{\downarrow}$, we get that (v_j, v_{j+1}) is such that either one of the following is true, either $\mu(v_j) \succ \mu(v_{j+1})$, or $\mu(v_j) = \mathsf{GCA}(\mu(v_j), \mu(v_{j+1}))$.

Therefore, t, defined as the minimum element that occurs infinitely often in the sequence $\mu(\pi)$ must be the greatest common ancestor of $\mu(v_i)$ and $\mu(v_{i+1})$ infinitely often. Moreover, it must be a common ancestor of $\mu(v_j)$ for almost all j.

To show 1 consider a vertex v_i, and (v_i, v_{i+1}) which occurs infinitely often in the play π for which $\mu(v_i) = t$ and also where the edge is consistent. This especially means that this edge satisfies condition $\mathsf{G}_{\succ}$ or $\mathsf{G}_{\downarrow}$. If v_i is coloured with $\perp$, this edge can only satisfy $\mathsf{G}_{\succ}$, and hence $\mu(v_i) \succ \mu(v_{i+1})$, a contradiction to the assumption that $\mu(v_i) = \min\inf(\mu(\pi))$.

Item 2, which claims that $c \in G_v$ infinitely often for vertices from the play π also follows from the above conditions as edge (v_i, v_{i+1}) identified in the above condition should satisfy $\mathsf{G}_{\downarrow}$infinitely often where $\mu(v_i) = t$.

Finally, we show Item 3 that $c \notin B_v$ for any v after some finite prefix of π. This is because for any (v_j, v_{j+1}), where we have $c \notin \mathsf{ColourSet}(\mu(v_j))$ from condition B. Since we had earlier observed that t is a common ancestor of $\mu(v_j)$ for all but finitely many of the edges (v_j, v_{j+1}) in π, we must have $c \notin B_{v_j}$ for all but finitely many v_js. □

Remark 7. *A similar statement to the equivalence of item 1 and 2 has been proved in the work of Klarlund and Kozen [KK91] for the restricted setting of Rabin graphs, however, a reader familiar with their work might have observed some differences in the definition of a measure. Our definition of colourful trees is more restrictive than theirs. For instance, colourful trees in the work of Klarlund and Kozen have no restrictions about the colours along a path in a tree, i.e, in their definition, the trees can have the same colour along a path and in fact only a partial colouring is required. However, an examination of their proof reveals that in the direction of the proof where they construct a Rabin measure, they inherently use a construction which produces a mapping into colourful trees as we have defined and therefore, it is enough to only consider such trees. We make this explicit and have proved it to suit our situation.*

9.3 Lifting algorithm for Rabin games

We wish to utilise the characterisation of Rabin games in terms of colourful decompositions and Rabin measures that map vertices to colourful trees to produce faster algorithms. We first provide an algorithm, that given a (c_0, C)-Rabin game decides if this game has a Rabin measure into a fixed (c_0, C)-colourful tree. This algorithm is a lifting algorithm that keeps track of an underlying map from the vertices to a labelled colourful tree and then repeatedly modifies this map until it obtains a Rabin measure. The running time of our algorithm is given in terms of the time it takes to navigate such a tree, but later in Section 9.4, we show the exact values of tree that are constructed to solve required to solve Rabin games.

A reader familiar with the work of Colcombet, Fijalkow, Gawrychowski, and Ohlmann [CFGO22] as well as our charecterisation can deduce that colourful trees form what are known as a "universal graph" of a Rabin game. Moreover, Colcombet, Fijalkow, Gawrychowski, and Ohlmann [CFGO22] show that if such a universal graph exists, then one can also construct a (progress measure) lifting algorithm that runs in time that is linear in the size of a colourful tree into which we have a Rabin measure for the game. However, implementing such an algorithm is non-trivial, as the proof of space requirements makes certain assumptions on the underlying model. We hope that our description provided here makes any future implementation straightforward, and this serves as a quick resource for such endeavours.

We describe an algorithm that identifies if a Rabin game $\mathcal{G}$ is winning for Steven, using Rabin measures defined earlier for Rabin graphs. Towards this goal, we define monotonic, inflationary operators on the set of all maps from vertices of a game to a tree such that the simultaneous fixpoints of these operators exactly correspond to a Rabin measure.

Consider a Rabin measure μ which is a function mapping the vertices V of a (c_0, C)-colourful Rabin game $\mathcal{G}$ into an $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$. We define a function lift_μ, which maps edges E of the arena of the game to $\mathcal{L}^\top$. For an edge $u \to v$ of $\mathcal{G}$, we define $\text{lift}_\mu(u, v)$ to be the smallest element t in $\mathcal{L}^\top$ such that $t \succeq \mu(u)$ and edge $u \to v$ is consistent with respect to the mapping $\mu[u := t]$. We use the notation $\mu[u := t]$ to indicate the mapping μ' where $\mu'(x) = \mu(x)$ if $x \neq u$ and $\mu'(x) = t$ if $x = u$.

For each vertex v, we define an operator Lift_v on the lattice of all maps from V to $\mathcal{L}^\top$. The operator Lift_v only modifies an input map μ at v and nowhere else. We define

$$\text{Lift}_v(\mu)(u) = \begin{cases} \mu(u) & \text{for } u \neq v \\ \min_{(v,w)\in E} \{\text{lift}_\mu(v, w)\} & \text{if } u = v \in V_c \\ \max_{(v,w)\in E} \{\text{lift}_\mu(v, w)\} & \text{if } u = v \in V_e. \end{cases}$$

Proposition 9.3.1. *The function* Lift_v *is monotone for each* v.

Proof. We show that for two measures $\mu_1 \preceq \mu_2$, $\text{Lift}_v(\mu_1) \sqsubseteq \text{Lift}_v(\mu_2)$. Note that it suffices to show that for Steven's (resp. Audrey's) vertices v, the value $t_1 = \min_{v\to w}\{\text{lift}_{\mu_1}(v, w)\}$ is at most as large as $t_2 = \min_{v\to w}\{\text{lift}_{\mu_2}(v, w)\}$ (using max for Audrey instead). Instead, we argue that μ_1', defined as $\mu_1[v := t_2]$ ensures that the vertex v is consistent. Let $v \to w$ be the edge that is consistent in $\mu_2' = \text{Lift}_v(\mu_2)$. We claim that the same edge(s) $v \to w$ is still consistent in μ_1'.

- If $v \to w$ satisfied $\mathsf{G}_\succ$ with respect to μ_2', then it continues to satisfy $\mathsf{G}_\succ$ with respect to μ_1', since $\mu_1'(v) = \mu_2'(v) \succ \mu_2(w) \succeq \mu_1'(w)$.

- If $v \to w$ satisfied $\mathsf{G}_\downarrow$ with respect to μ_2', then it either continues to satisfy $\mathsf{G}_\downarrow$, or satisfies $\mathsf{G}_\succ$ with respect to μ_2'. To see this, we observe that $\mu_2'(v)$ is an ancestor of $\mu_2'(w) = \mu_2(w)$, and $\mu_2'(w) \succeq \mu_1'(w)$. From Proposition 9.2.4, we consequently get $\mu_1'(v) \succ \mu_1'(w)$ or $\mu_1(v)$ is an ancestor of $\mu_1'(w)$, which is exactly $\mathsf{G}_\succ$ or $\mathsf{G}_\downarrow$respectively.

- If $u \to v$ satisfied B with respect to μ_2', then it continues to satisfy B with respect to μ_1', since $\mu_2'(v) = \mu_1'(v)$ and $\mathsf{ColourSet}(\mu(u)) \cap B_u = \mathsf{ColourSet}(\mu_1(u)) \cap B_u = \varnothing$. □

We know that each Lift_v is inflationary and monotone. Therefore, the simultaneous least fixpoint of Lift_v on the map μ, which maps all vertices to the root of $\mathcal{L}$ exists (from the Knaster-Tarski theorem [Tar55]). We can moreover state the following proposition that such fixpoints correspond to the Rabin measures, which almost follows from our definitions.

Proposition 9.3.2. *For a (c_0, C)-colourful Rabin game $\mathcal{G}$ where the vertex set is V and a fixed $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$,*

- *any simultaneous fixpoint of the set of functions Lift_v for all $v \in V$ is a Rabin measure;*

- *any Rabin measure is a simultaneous fixpoint of Lift_v for all $v \in V$.*

Our algorithm, like any other progress-measure algorithm, computes this simultaneous fixpoint of Lift as follows, the correctness of which follows from Propositions 9.3.1 and 9.3.2.

Algorithm 10 The lifting algorithm on game (c_0, C)-colourful Rabin game $\mathcal{G}$ with vertices V to tree $\mathcal{L}$

1: **Initialise:** For each $v \in V$, $\mu(v)$ is declared to be root in $\mathcal{L}$
2: **while** there is some vertex v that is inconsistent with respect to μ. **do**
3: $\quad \mu \leftarrow \mathrm{Lift}_v(\mu)$.
4: **end while**
5: **return** μ

Remark 8. *If there is (c_0, C)-colourful Rabin game $\mathcal{G}$ and a $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}'$, such that there is a Rabin measure μ' from V to $\mathcal{L}'$, and $\mathcal{L}$ embeds $\mathcal{L}'$, then there is also Rabin measure μ to $\mathcal{L}$. All the elements that are not mapped to $\top$ by μ' are still not mapped to $\top$ by μ.*

Running time complexity. For a finer analysis of the runtime, we need to understand the size of the lattice where the lifting algorithm takes place. However, in this section, we restrict ourselves to analysing the running time of our algorithm for a fixed tree $\mathcal{L}$, whose size is denoted by $|\mathcal{L}|$. Additionally, we report our runtimes in the form of operations performed to navigate the underlying tree $\mathcal{L}$. In Section 9.4, we construct sufficiently large colourful trees that can solve Rabin games of a fixed number of vertices and colours and expand on the time and space complexity of these operations on these constructed trees.

Lemma 9.3.3. *Given a mapping from the vertices of an n-vertex (c_0, C)-colourful Rabin game $\mathcal{G}$ to a $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$, the value of $\mathrm{Lift}_v(\mu)(v)$ can be computed in time proportional to $\mathcal{O}(\deg(v) \cdot T_{\mathsf{next}})$, where $\deg(v)$ is the degree (number of outgoing edges) of v and T_{next} is defined as the maximum of*

- *the time taken to make a linear pass on a node in $\mathcal{L}$;*
- *the time taken to compute the next node in $\mathcal{L}$;*
- *given $t \in \mathcal{L}$ and $C' \subseteq C$ such that $\mathsf{colour}(t) \in C'$, the time taken to find the next node that uses colours only from $C' \cup \{\bot\}$.*

Proof. We first answer the following question: given an edge $u \to v$ and a mapping μ to $\mathcal{L}^\top$, can we calculate $\mathrm{lift}_\mu(u, v)$ *quickly*?

We show how to define and compute $\mathrm{lift}_\mu(u, v)$ function using the following subroutines:

- computing the next node, and
- computing the next node whose colour set contains colours only from $C' \cup \{\bot\}$ where the colouring of the given node is in $C' \subseteq C$.

Henceforth, we denote the successor of node t in $\mathcal{L}^\top$ with respect to the order $\prec$ by $\mathsf{next}(t)$. A naive way to compute $\mathrm{lift}_\mu(u, v)$ would be to apply next to $\mu(u)$ and to check each time if the edge $u \to v$ satisfies the consistency properties. But such an algorithm would potentially take exponential time to compute some lift functions. We remark however that this naive algorithm would only add a polynomial factor to the upper bound to the worst-case complexity of our run-time after amortisation.

We will now give the function, which directly computes $\mathrm{lift}_\mu(u, v)$ using only two primitives after a linear scan

(a) computing the next node in the tree;

(b) computing the next node whose colour set contains colours only from $C' \cup \{\bot\}$ where the colouring of the given node is in $C' \subseteq C$.

It can be inferred from the procedure described in the following paragraphs that we need only finitely many linear passes on a node when represented as a sequence of labels and colours.

Edge $u \to v$ is already consistent. In this case, $u \to v$ already satisfies at least one of $\mathsf{G}_\succ$ or $\mathsf{G}_\downarrow$ along with (R) in μ. Hence $\mathrm{lift}_\mu(u, v)$ is set to $\mu(u)$, continues to make $u \to v$ consistent.

Edge $u \to v$ satisfies $\mathsf{G}_\succ$ but not B. In this case, we only need to find the smallest value larger than $\mu(u)$ whose colour set does not contain any colours from B_u. Let $\mu(u) = \langle \alpha_1 c_{i_1}, \ldots, \alpha_m c_{i_m} \rangle$ where $\alpha_i \in \mathbb{L}$. We achieve this by finding the largest position s that gives $\mathsf{ColourSet}(\langle \alpha_1 c_{i_1}, \ldots, \alpha_s c_{i_s} \rangle) \cap B_u = \varnothing$. Then we compute the smallest child t larger than the node above, $t = \langle \alpha_1 c_{i_1}, \ldots, \alpha_{s+1} c_{i_{s+1}} \rangle$ that gives $\mathsf{ColourSet}(t) \cap B_u = \varnothing$ and set $\mathrm{lift}_\mu(u, v)$ to t. The computation clearly takes time at most T_{next}.

Since $\mu(u) \succ \mu(v)$, $\mathrm{lift}_\mu(u, v) \succ \mu(u)$ and $\mathrm{lift}_\mu(u, v)$ doesn't use any colours from B_v, the edge $u \to v$ satisfies $\mathsf{G}_\succ$ and B in the new mapping.

Edge $u \to v$ satisfies $\mathsf{G}_\downarrow$ but not $\mathsf{G}_\succ$ or B. We again take $\mu(u) = \langle \alpha_1 c_{i_1}, \ldots, \alpha_m c_{i_m} \rangle$. Since $u \to v$ satisfies $\mathsf{G}_\downarrow$, we know that $\mu(u)$ is an ancestor of $\mu(v)$. We argue that the smallest value larger than $\mu(u)$ that also satisfies B does not satisfy $\mathsf{G}_\downarrow$, but rather satisfies $\mathsf{G}_\succ$. This is because there is an ancestor of $\mu(u)$ (and thus, of $\mu(v)$) that is coloured by a bad colour for u. Since $\mathrm{lift}_\mu(u, v)$ must be larger than $\mu(u)$, it cannot be set to an ancestor of $\mu(u)$. Then, it should be set to a larger sibling of one of the ancestors of $\mu(u)$. Since any larger sibling of an ancestor of $\mu(v)$ is always larger than $\mu(v)$, the smallest value of $\mathrm{lift}_\mu(u, v)$ that makes $u \to v$ consistent satisfies $\mathsf{G}_\succ$. We have therefore reduced this case to the previous one.

Edge $u \to v$ satisfies neither $\mathsf{G}_\succ$, $\mathsf{G}_\downarrow$ or B. Since the edge does not satisfy $\mathsf{G}_\succ$, we know $\mu(u) \preceq \mu(v)$. We go through the ancestors of $\mu(v)$ one by one in increasing order to see if there exists one that is both strictly larger than $\mu(u)$, and satisfies B. If there exists one, then we set $\mathrm{lift}_\mu(u, v)$ to the first such value found, and $u \to v$ satisfies $\mathsf{G}_\downarrow$ and B in the new mapping. This computation takes a linear scan through at most the length of $\mu(v)$. If none of the ancestors satisfy these constraints, then we know that $\mathrm{lift}_\mu(u, v)$ has to be at least as large as $\mathsf{next}(\mu(v))$.

Thus $u \to v$ has to satisfy $\mathsf{G}_{\succ}$ and B in the next mapping. Once more, we have reduced this case to the previous ones.

We have concluded that computing $\mathrm{lift}_\mu(v, w)$ takes time at most $\mathcal{O}(T_{\mathsf{next}})$. Recall the definition of Lift_v by using $\mathrm{lift}_\mu(v, w)$ as a subroutine.

$$\mathrm{Lift}_v(\mu)(u) = \begin{cases} \mu(u) & \text{for } u \neq v \\ \min_{(v,w)\in E} \{\mathrm{lift}_\mu(v, w)\} & \text{if } u = v \in V_S \\ \max_{(v,w)\in E} \{\mathrm{lift}_\mu(v, w)\} & \text{if } u = v \in V_A \end{cases}$$

It is therefore easy to conclude that $\mathrm{Lift}_v(\mu)(v)$ takes time at most $\mathcal{O}(\deg(v) \cdot T_{\mathsf{next}})$ □

First, we observe that performing Lift_v on the mapping strictly increases the mapping for a vertex that has no consistent edges. Each operation of Lift_v also calls at most $\deg(v)$ many calls of $\mathrm{lift}_\mu(v, u)$ for some edge $v \to u$. Suppose each operation $\mathrm{lift}_\mu(v)$ takes time T_{next}, to find the value of $\mathrm{Lift}_v(\mu)(v)$ takes time at most $\deg(v) \cdot T_{\mathsf{next}}$. Since each non-trivial application of Lift_v strictly increases the value that v is mapped to, it can be called at most as many times as the number of nodes in tree $\mathcal{L}$, this ensures that the time taken is

$$\sum_{v\in V} \deg(v)|\mathcal{L}|\,(T_{\mathsf{next}}) \in \mathcal{O}(m|\mathcal{L}|T_{\mathsf{next}})\,.$$

We finally conclude this section with the following theorem.

Theorem 9.3.4. *For a (c_0, C)-colourful Rabin game $\mathcal{G}$ with n vertices and m edges, and a $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$, Algorithm 10 on game $\mathcal{G}$ with colourful tree $\mathcal{L}$ returns the smallest Rabin measure to $\mathcal{L}^\top$ in time $\mathcal{O}(m|\mathcal{L}|T_{\mathsf{next}})$ where T_{next} is as defined in Lemma 9.3.3 and $|\mathcal{L}|$ denotes the number of nodes in $\mathcal{L}$.*

9.4 Small colourful-universal trees

We observed that our algorithm from the previous section correctly identifies the smallest Rabin measure into a fixed labelled colourful tree $\mathcal{L}$. However, by Theorem 9.2.1, we know that *there exists* an $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$ with at most n leaves. Therefore, for a Rabin game, there is a Rabin measure into $\mathcal{L}^\top$ where all start vertices from which the game is winning for Steven are not mapped to $\top$. To successfully determine the winner of all (c_0, C)-colourful Rabin games with n vertices, we need to ensure that the tree $\mathcal{L}$ used in Algorithm 10 would be able to

embed all (c_0, C)-colourful trees with n leaves. Since the running time is linearly dependent on the size of the tree, smaller trees that satisfy the above property are desirable.

We show how to construct *universal* colourful trees. We say that (c_0, C)-colourful n-universal trees are colourful trees that are large enough to embed *any* (c_0, C)-colourful $\mathcal{L}$ with n-nodes. We also modify previously discussed Jurdziński-Lazić universal trees [JL17] to encode each node of these universal colourful trees using space proportional to a $\mathcal{O}(k \log k \log n)$, which helps navigate these labelled colourful trees efficiently.

Definition 9.4.1 (n-Universal (c_0, C)-colourful tree)**.** *A (c_0, C)-colourful tree $\mathcal{U}$ is* n-universal*, if it embeds any (c_0, C)-colourful tree $\mathcal{T}$ with at most n leaves.*

We henceforth assume that $C = \{c_1, \ldots, c_h\}$, with the fixed ordering $c_1 < c_2 < \cdots < c_h$ on the colours, and $k = h + 1$.

A straightforward approach to creating an n-universal (c_0, C)-colourful tree could involve combining all potential (c_0, C)-colourful trees with no more than n leaves and with the root colour c_0, and concatenating them. Clearly, such a finite n-Universal (c_0, C)-colourful tree can be created as there are only finitely many such trees up to isomorphism (for a fixed C and n). But of course, this tree is not only large, but can also be difficult to navigate. An alternative, more manageable approach is to construct a tree that branches $n \cdot |C|$ many times at the root. These initial subtrees, resulting from this $n \cdot |C|$ branching, are rooted with n repeating blocks of the h colours $c_1, c_2, \ldots, c_h$ in order. For example, if h were 2, then the colours of the children of the root would be, in that order, $c_1, c_2, c_1, c_2, \ldots, c_1, c_2$. Each of the children in turn branches into $n \cdot (|C| - 1)$ many times recursively. This creates a tree of size bounded by $n^h h!$. Remarkably, this very tree (whose nodes are represented as tuples) serves as the underlying structure that underpins the algorithm devised by Piterman and Pnueli [PP06] which led to their $\mathcal{O}(mn^{k+1}kk!)$ algorithm.

In the subsequent text, we give a more involved construction—inspired by the Jurdziński-Lazić universal trees—of a significantly smaller universal tree. In our construction, we inductively describe such a (c_0, C)-colourful n-universal tree, which we call $\mathcal{U}^{\ell}_{(c_0,C)}$, for all $n \leqslant 2^{\ell}$.

- if $C = \varnothing$, then there is exactly one tree to embed, and therefore

$$\mathcal{U}^{\ell}_{(c_0,C)} = \left(c_0, \left\langle (\bot, \langle\rangle)^{2^{\ell}} \right\rangle\right)$$

- if $\ell = 0$, then the tree to be embedded has exactly one leaf and therefore, for each colour c_i in C, we have a child of colour c_i which hosts a subtree rooted at c_i. This is defined inductively as

$$\mathcal{U}^0_{(c_0,C)} = \left(c_0, \left\langle \mathcal{U}^0_{(c_1,C_1)}, \ldots, \mathcal{U}^0_{(c_h,C_h)}, (\bot, \langle\rangle)\right\rangle\right)$$

  where C_i is $C \setminus \{c_i\}$.

- if $C \neq \varnothing$ and $\ell > 0$, then we define the coloured tree to be two copies of an $n/2$-universal tree, and $|C|$ many copies of the n-universal tree where one colour is dropped each time. More formally,

$$\mathcal{U}^\ell_{(c_0,C)} = \mathcal{U}^{\ell-1}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{U}^\ell_{(c_1,C_1)}, \ldots, \mathcal{U}^\ell_{(c_h,C_h)}, (\bot, \langle\rangle)\right\rangle\right) \cdot \mathcal{U}^{\ell-1}_{(c_0,C)}.$$

In Fig. 9.6, we demonstrate how the inductive construction is done if $c_0 = \bullet$ and the set of colours is $C = \{\bullet, \bullet, \bullet\}$. To the left and right are the $(\bullet, C)$-colorful $n/2$-universal trees and between them, there are $|C|$ many n-universal trees each of which uses one fewer colour and one node with just the dummy colour represented there by $\otimes$.

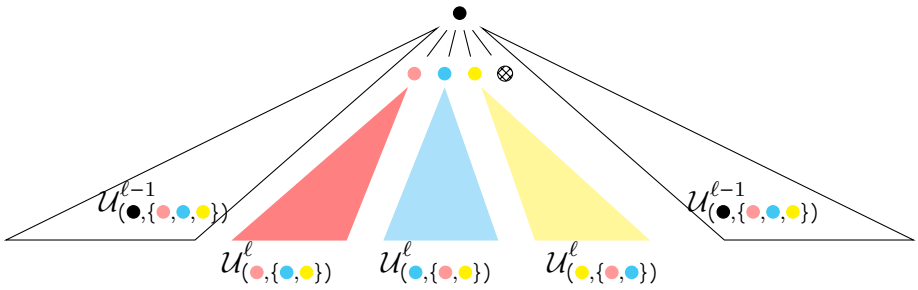


Figure 9.6: Inductive construction of a colourful n-universal tree

Theorem 9.4.2. *For $C \neq \varnothing$, and $k = |C| + 1$, and $\ell = \lceil \lg n \rceil$, the colourful tree $\mathcal{U}^\ell_{(c_0,C)}$ constructed is a (c_0, C)-colourful n-universal tree with the number of leaves no larger than*

$$nk! \left(\min \left\{ n2^k, \binom{\ell + k}{k-1} \right\} \right).$$

Proof. Firstly, we show that $\mathcal{U}^\ell_{(c_0,C)}$ is (c_0, C)-colourful n-universal tree (in Proposition 9.4.3). Then we prove that $\mathcal{U}^\ell_{(c_0,C)}$ has at most $2^k \cdot k! \cdot 4^\ell$ leaves in Lemma 9.4.4 and at most $\binom{\ell+k}{k-1} \cdot 2^\ell \cdot k!$ leaves in Lemma 9.4.5, both proved by induction, leading to the proof of our theorem. □

Proposition 9.4.3. *The (c_0, C)-colourful tree $\mathcal{U}^{\ell}_{(c_0,C)}$, embeds any (c_0, C)-colourful tree with at most n leaves where $\ell \geqslant \lceil \log n \rceil$.*

Proof of Proposition 9.4.3. Consider any (c_0, C)-colourful tree $\mathcal{T}$ with n leaves. The statement is trivial if $C = \varnothing$, since from our construction, our tree is such that all n leaves have colour $\bot$. We assume $C \neq \varnothing$ but $\ell = 0$, and therefore $n = 1$. Let $C = \{c_1, \ldots, c_h\}$. In this case, we have

$$\mathcal{U}^{0}_{(c_0,C)} = \left(c_0, \left\langle \mathcal{U}^{0}_{(c_1,C_1)}, \ldots, \mathcal{U}^{0}_{(c_h,C_h)}, (\bot, \langle\rangle) \right\rangle\right)$$

We must either have $\mathcal{T} = (c_0, \langle \mathcal{T}_i \rangle)$ for some $(c_i, C \setminus \{c_i\})$-colourful tree $\mathcal{T}_i$ or alternatively, $\mathcal{T} = \langle (\bot, \langle\rangle) \rangle$, and clearly from the construction, it follows that this tree can be embedded in $\mathcal{U}^{0}_{(c_0,C)}$, recursively, by choosing an appropriate subtree $\left(c_i, \mathcal{U}^{0}_{(c_i,C_i)}\right)$, and recursively embedding $\mathcal{T}_i$ in $\mathcal{U}^{0}_{(c_i,C_i)}$.

If we consider the case where $n > 1$ (and therefore $\ell > 0$), and for this case, let $\mathcal{T} = (c_0, \langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle)$. Let n_p represent the number of leaves of $\mathcal{T}_p$. We know $\sum n_p = n$. For each p, we define

$$\mathcal{T}_{<p} = \left(c_0, \left\langle \mathcal{T}_1, \ldots, \mathcal{T}_{p-1} \right\rangle\right) \qquad \text{and} \qquad \mathcal{T}_{>p} = \left(c_0, \left\langle \mathcal{T}_{p+1}, \ldots, \mathcal{T}_m \right\rangle\right).$$

There must be at least one p for which both trees $\mathcal{T}_{<p}$ as well as $\mathcal{T}_{>p}$ have size at most $n/2$. The existence of such a p can be shown by defining the summation $N_j = \sum_{i=1}^{j} n_i$ which ranges from 0 to n as j ranges from 1 to m. Then there must be some point where N_j exceeds $n/2$, giving us our desired p.

Since both $\mathcal{T}_{<p}$ and $\mathcal{T}_{>p}$ have at most $n/2$ leaves, by induction $\mathcal{U}^{\ell-1}_{(c_0,C)}$ embeds $\mathcal{T}_{<p}$ as well as $\mathcal{T}_{>p}$, since C contains all the colours in $\mathcal{T}_{<p}$ and $\mathcal{T}_{>p}$ and each tree has less than $n/2$ leaves. Furthermore, $\mathcal{U}^{\ell}_{(c_{i_p},C_{i_p})}$ embeds $\mathcal{T}_p$, where c_{i_p} is the colour of the root of $\mathcal{T}_p$ and $C_{i_p} = C \setminus \{c_{i_p}\}$. Observe that for each c_i, there is a copy of the tree of $\mathcal{U}^{\ell}_{(c_i,C_i)}$, where $C_i = C \setminus \{c_i\}$. Hence from the construction of $\mathcal{U}^{\ell}_{(c_0,C)}$, the tree

$$\mathcal{T} = \mathcal{T}_{<p} \cdot \left(c_0, \left\langle \mathcal{T}_p \right\rangle\right) \cdot \mathcal{T}_{>p}$$

can be embedded into

$$\mathcal{U}^{\ell}_{(c_0,C)} = \mathcal{U}^{\ell-1}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{U}^{\ell}_{(c_1,C_1)}, \ldots, \mathcal{U}^{\ell}_{(c_h,C_h)}, (\bot, \langle\rangle) \right\rangle\right) \cdot \mathcal{U}^{\ell-1}_{(c_0,C)}. \qquad \square$$

Lemma 9.4.4. *The tree $\mathcal{U}^{\ell}_{(c_0,C)}$ has at most $2^k \cdot k! \cdot 4^{\ell}$ many leaves where $k = |C \cup \{c_0\}|$.*

Proof. Let us denote by $U(\ell, h)$, the number of leaves in the tree $\mathcal{U}^{\ell}_{(c_0,C)}$ defined

above, where $|C| = h$.

If $k = 1$, $h = k - 1 = 0$ then $U(\ell, h) = 2^\ell$ by construction.

If $\ell = 0$, then we show by induction a stronger statement that $U(0, h) \leqslant h!h$ for all values of $h \geqslant 1$. Indeed,

$$\mathcal{U}^0_{(c_0,C)} = \left(c_0, \left\langle \mathcal{U}^0_{(c_1,C_1)}, \ldots, \mathcal{U}^0_{(c_h,C_h)}, (\bot, \langle\rangle)\right\rangle\right)$$

From this we can infer that

$$U(0, h) \leqslant (k-1) \cdot U(0, k-1) + 1$$

Since we already know $U(0, 1) = 1$, inductively, we can show that

$$U(0, h) \leqslant h \cdot U(0, h-1) + 1 \leqslant h \cdot ((h-1)!) + 1 \leqslant h!h$$

For $\ell, h > 0$, recall that

$$\mathcal{U}^\ell_{(c_0,C)} = \mathcal{U}^{\ell-1}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{U}^\ell_{(c_1,C_1)}, \ldots, \mathcal{U}^\ell_{(c_h,C_h)}, (\bot, \langle\rangle)\right\rangle\right) \cdot \mathcal{U}^{\ell-1}_{(c_0,C)}.$$

Therefore, we see that for $\ell, h > 0$, the following recurrence relation holds

$$U(\ell, h) = 2 \cdot U(\ell - 1, h) + h \cdot U(\ell, h-1) + 1$$

We prove $U(\ell, h) \leqslant 4^\ell \cdot hh! \cdot 2^h$, by induction.

For the base case, for the values $U(0, h)$ and $U(\ell, 0)$ are at most $4^\ell \cdot hh! \cdot 2^h$, and therefore the inequality holds. We assume, for $t < \ell$ and $j < h$, that $U(t, j) \leqslant 4^t \cdot 2^j \cdot jj!$ as our induction hypothesis. For this ℓ and h, observe

$$\begin{aligned}
U(\ell, h) &= 2U(\ell - 1, h) + hU(\ell, h-1) + 1 \\
&\leqslant 2 \cdot \left((4)^{\ell-1} \cdot 2^h \cdot (h)!h\right) + h \cdot \left(4^\ell \cdot 2^{h-1} \cdot h!\right) + 1 \\
&\leqslant \frac{1}{2}\left(4^\ell \cdot 2^h \cdot h!h\right) + h\left(4^\ell \cdot 2^{h-1} \cdot (h-1)!(h-1)\right) + \left(4^\ell \cdot 2^{h-1} \cdot (h)!\right) \\
&= \frac{1}{2}\left(4^\ell \cdot 2^{h-1} \cdot h!h\right) + \left(4^\ell \cdot 2^{h-1} \cdot h!h\right) \\
&= \frac{1}{2}\left(4^\ell \cdot 2^{h-1} \cdot h!h\right) + \frac{1}{2} \cdot \left(4^\ell \cdot 2^h \cdot h!h\right) \\
&= 4^\ell \cdot 2^h \cdot (h)!h
\end{aligned}$$

Since $h = k - 1$, our claim follows. □

Lemma 9.4.5. *The tree* $\mathcal{U}^\ell_{(c_0,C)}$ *has size at most* $\binom{\ell+k}{k-1} \cdot 2^\ell \cdot k!$*, where* $k = |C \cup \{c_0\}|$.

Proof. Let us again denote by $U(\ell, h)$, the number of leaves in the tree $\mathcal{U}^{\ell}_{(c_0,C)}$, where $|C| = h = k - 1$.

If $h = 0$, then $U(\ell, h) = 2^{\ell}$ by construction, and therefore we have $U(\ell, 0) = \binom{\ell+h+1}{h} h! 2^{\ell}$.

If $\ell = 0$, recall from the proof of Lemma 9.4.4 we show that $U(0, h) \leqslant h!h$ for all values of $h \geqslant 0$.

Now, suppose $\ell, h > 0$, then we have

$$\begin{aligned}
U(\ell, h) &= 2U(\ell - 1, h) + hU(\ell, h - 1) + 1 \\
&\leqslant 2 \cdot \left((2)^{\ell-1} \cdot \binom{\ell + h}{h} (h)!h \right) + h \cdot \left(2^{\ell} \cdot \binom{\ell + h}{h - 1} (h - 1)!(h - 1) \right) + 1 \\
&\leqslant \left(2^{\ell} \cdot \binom{\ell + h}{h} (h)!h \right) + \left(2^{\ell} \cdot \binom{\ell + h}{h - 1} (h)!(h - 1) \right) + \left(2^{\ell} \cdot \binom{\ell + h}{h - 1} (h)! \right) \\
&\leqslant 2^{\ell} \cdot h!h \left(\binom{\ell + h}{h} + \binom{\ell + h}{h - 1} \right) \\
&= 2^{\ell} \cdot h!h \binom{\ell + h + 1}{h} \square
\end{aligned}$$

9.4.1 Lower bounds on the size of universal colourful trees

We show that our construction is near-optimal as we have a lower bound on the size of n-universal (c_0, C)-colourful trees, which is within a polynomial factor of the upper bound.

Lemma 9.4.6 (Lower bound on universal colourful trees)**.** *Any n-universal (c_0, C)-colourful tree must have size at least $\binom{\ell+k-1}{\ell}(k-1)!$ where $k = |C|+1$, and $\ell = \lfloor \log n \rfloor$.*

Proof. We first recall a theorem on the lower bound of (colourless) universal trees by Czerwiński et al. [CDF$^+$19], inspired by the lower bound results for trees by Goldberg and Livshits [GL68].

Theorem 9.4.7 ([CDF$^+$19], Theorem 2.3)**.** *For natural numbers n and h, every (n, h)-universal tree has at least $\binom{\lfloor \lg n \rfloor + h - 1}{h-1}$ leaves.*

To prove our results, we first fix a permutation of colours $c_{i_1}, \ldots, c_{i_h}$ and consider any tree with n leaves where the order of colours from the root to the leaf is exactly the same as the given permutation. We also require that the tree be equitable, that is, all leaves have the same depth from the root. Then this tree must have size at least the size of a 2^{ℓ}-universal tree of height h (defined for ordered tree without colours). Such universal trees have size at least $\binom{\ell+h-1}{h-1}$ from

Theorem 9.4.7. Then, for each choice of permutation, the universal tree restricted to that permutation must have size $\binom{\ell+h-1}{h-1}$. Furthermore, two universal trees obtained by fixing different permutations cannot share a leaf due to the distinct colours assigned to the same ancestor of the leaf. Therefore, we get a lower bound of $\binom{\ell+h-1}{h-1}h!$ on the size of (c_0, C)-colourful n-universal trees.

Immediately, for $k = |C| + 1$, this gives us the bound $\binom{\ell+k-2}{\ell}(k-1)!$. This closely matches one of the upper bound of our construction by at most a polynomial factor in n and k. □

Labelled universal colourful trees. Here, we give a labelling of a universal colourful tree described in the previous section by giving an $\mathbb{W}$-labelling of any (c_0, C)-colourful tree. Recall from Chapter 2 that the set $\mathbb{W} = \{0,1\}^*$ has a bit-string ordering on $\mathbb{W}$ which is a total ordering. $0 < \varepsilon < 1$ and for $b_1, b_2 \in \{0,1\}$ we have $b_1 \cdot w_1 < b_2 \cdot w_2$ if and only if $b_1 < b_2$ or $b_1 = b_2$ and $w_1 < w_2$.

Any node in a $\mathbb{W}$-labelled (c_0, C)-colourful tree can be represented by a word of the form below

$$t = \{0,1\}^* c_{i_1} \cdot \{0,1\}^* c_{i_2} \cdot \ldots \cdot \{0,1\}^* c_{i_m},$$

where $c_{i_j} \neq c_{i_k}$ if $i \neq j$ and $c_{i_j} = \perp$ if and only if $j = m$. We call the number of 0s and 1s occurring in t, *the number of bits used to label* t. We show in the following lemma that it is possible to have a labelling of our universal colourful tree $\mathcal{U}^{\ell}_{(c_0,C)}$ such that the encoding of each node in it is "short".

Lemma 9.4.8. *There is a $\mathbb{W}$-labelling of the tree $\mathcal{U}^{\ell}_{(c_0,C)}$, denoted by $\mathcal{L}^{\ell}_{C}$ such that the number of bits used to label any node of $\mathcal{L}^{\ell}_{C}$ is at most ℓ.*

Proof. We expand on the notation that we used in Chapter 5 in the construction below to obtain a $\mathbb{W}$-labelling of $\mathcal{U}^{\ell}_{(c_0,C)}$, and recall some new notation.

- For $b \in \{0,1\}$ and each $\omega_i \in \mathbb{W} \cdot C$, we define

  $$[\,\mathcal{L}\,]^b = \{(b \cdot \omega_1, \omega_2, \ldots, \omega_m) \mid (\omega_1, \omega_2, \ldots, \omega_m) \in \mathcal{L}\}$$

  In words, $[\mathcal{L}]^b$ is the labelled ordered tree that is obtained from $\mathcal{L}$ by adding an extra copy of bit b as the leading bit in the labels of all children of the root of $\mathcal{L}$.

- Recall that for a prefix closed set $\mathcal{L}$ and $\omega \in \mathbb{W} \cdot C$, we define $\omega \odot \mathcal{L}$ as the

prefix-closure of the set

$$\{(\omega, \omega_1, \omega_2, \ldots \omega_m) \mid (\omega_1, \omega_2, \ldots, \omega_m) \in \mathcal{L}\}$$

obtained by the addition of the term ω to the prefix of the tuples in $\mathcal{L}$ for each tuple in the set $\mathcal{L}$, and taking its prefix closure.

Consider the (c_0, C)-colourful 2^ℓ-universal tree $\mathcal{U}^\ell_{(c_0,C)}$.

- if $\ell = 0$ and $C = \varnothing$, then clearly, $\mathcal{L}^\ell_C$, defined as the empty tuple $()$ and uses no bits to label each node in the tree.
- if $\ell = 0$ and $C \neq \varnothing$, then we define $\mathcal{L}^\ell_C$ to be

$$\bigcup_i \left(\varepsilon c_i \odot \mathcal{L}^0_{C_i}\right)$$

where each $\mathcal{L}^0_{C_i}$ is the recursively obtained labelling for $\mathcal{U}^0_{(c_i,C_i)}$. Observe that no extra bits are used in addition to the bits used by each $\mathcal{L}^0_{C_i}$. Since each $\mathcal{L}^0_{C_i}$ uses 0 bits to label their nodes, $\mathcal{L}^\ell_C$ also uses 0 bits to label each node in the tree. Also note that this set is prefix closed.

- if $\ell > 0$ and $C \neq \varnothing$ and recall that

$$\mathcal{U}^\ell_{(c_0,C)} = \mathcal{U}^{\ell-1}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{U}^\ell_{(c_1,C_1)}, \ldots, \mathcal{U}^\ell_{(c_h,C_h)}, (\bot, \langle\rangle)\right\rangle\right) \cdot \mathcal{U}^{\ell-1}_{(c_0,C)}.$$

Let $\mathcal{L}^\ell_C$ be a labelling of $\mathcal{U}^\ell_{(c_0,C)}$, defined as the prefix-closed set

$$\left[\, \mathcal{L}^{\ell-1}_C \,\right]^0 \cup \bigcup_i (\varepsilon c_i) \odot \mathcal{L}^\ell_{C_i}, \cup\, (\varepsilon \bot) \cup \left[\, \mathcal{L}^{\ell-1}_C \,\right]^1$$

where $\mathcal{L}^{\ell-1}_C$ and $\mathcal{L}^\ell_{C_i}$ are labellings of $\mathcal{U}^{\ell-1}_{(c_0,C)}$ and $\mathcal{U}^\ell_{(c_i,C_i)}$ respectively, and use at most $\ell - 1$ and ℓ bits to encode each of their nodes. Hence $\mathcal{L}^\ell_C$ as constructed uses at most ℓ bits to encode each node.

□

Recall that we denoted the time taken to navigate this tree for the purposes of our lifting algorithm as T_{next}. We rigorously prove that this value T_{next} for this tree defined above is $\mathcal{O}(k \log(k)\ell)$ where the tree uses k colours. Finally, we get our main theorem of the section as stated below. We postpone the technical details of the proof of the theorem to the end of this section.

Theorem G. *A winning strategy for Steven in a Rabin game with n vertices, m edges, and k colours can found using $\mathcal{O}(nk \log k \log n)$ space and time*

$$\widetilde{\mathcal{O}}\left(nm \cdot k! \min\left\{n2^k, \binom{\lceil \lg n \rceil + k}{k-1}\right\}\right).$$

Proof. We know that the lifting Algorithm 10 for a (c_0, C)-colourful tree finds the Rabin measure into the tree $\mathcal{L}$ in time $\mathcal{O}(mn|\mathcal{L}|T_{\mathsf{next}})$.

For a game with n vertices, we instantiate the algorithm with $\mathcal{L}$ being the $\mathbb{W}$-labelling of the (c_0, C)-colourful 2^ℓ-universal, tree the tree $\mathcal{U}^\ell_{(c_0,C)}$ constructed, where $\ell = \lceil \lg(n) \rceil$. $\mathcal{L}$ therefore has at most $\left(nk! \min\left\{n2^k, \binom{\lceil \lg n \rceil + k}{k-1}\right\}\right)$ many leaves from Theorem 9.4.2, and hence at most k times as many nodes. Moreover, we show that the time taken to navigate the tree T_{next} is at most $\mathcal{O}(k\ell \log k)$ in Lemma 9.4.9 and Proposition 9.4.10. □

The following lemma proves the last piece required in the proof of our theorem, by showing that one can navigate these trees quickly.

Lemma 9.4.9. *Given a node in the $\mathbb{W}$-labelled (c_0, C)-colourful tree $\mathcal{L}^\ell_C$, with at most 2^ℓ leaves one can compute the next node larger than a given node in time $\mathcal{O}(k \log(k)\ell)$, where $k = |\{c_0\} \cup C|$.*

Proof. We first introduce, for $a \in \mathbb{N}$, a function $\mathsf{nextstring}^a$ that takes a string ω on $\{0,1\}^*$ with $|\omega| \leqslant a$ and calculates the smallest ω' with $|\omega'| \leqslant a$ that is larger than ω, if it exists (with respect to the ordering on $\mathbb{W}$).

For example, for $a = 3$, the succinct encoding gives us the following order:

$$000 < 00 < 001 < 0 < 010 < 01 < 011 < \epsilon < 100 < 10 < 101 < 1 < 110 < 111$$

and the $\mathsf{nextstring}^a$ function gives us exactly this ordering. That is for instance, $\mathsf{nextstring}^3(0) = 010$ and $\mathsf{nextstring}^3(011) = \epsilon$. Additionally, for a newly introduced element $\lightning$, we set $\mathsf{nextstring}^a(1^a) := \lightning$, for example, $\mathsf{nextstring}^3(111) = \lightning$.

Let $\omega \in \{0,1\}^*$ with $|\omega| \leqslant b$. Then $\mathsf{nextstring}^a(\omega)$ is computed as follows,

- If $|\omega| < a$, then $\mathsf{nextstring}^a(\omega) = \omega 10^{a-1}$,
- If $|\omega| = a$,
  - If $\omega = \omega' 01^k$ for some ω' and $k \geqslant 0$, then $\mathsf{nextstring}^a(\omega) = \omega'$,
  - If $\omega = 1^a$, then $\mathsf{nextstring}^a(\omega) = \lightning$.

Next we define our desired function $\mathsf{next}^{\ell}_{C}(t)$ that takes a node of $\mathcal{L}^{\ell}_{C}$ and sends it to the next node that is is larger than t, and contains colours from the set C. If no such node exists, it sends it to $\top$.

We apply the following rules to calculate $\mathsf{next}^{\ell}_{C}(t)$ for some node $t = \langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m} \rangle$:

- If $c_{i_m} \neq \bot$, then t is not a leaf and therefore, $\mathsf{next}^{\ell}_{C}(t)$ is t's smallest child. $\mathsf{next}^{\ell}_{C}(t) = \langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m}, 0^a c \rangle$ where c is the minimum colour in $C^{\bot} \setminus \{c_{i_1}, \ldots, c_{i_m}\}$ and $a = \ell - \sum_{i=1}^{m} |\omega_i|$.
- If $c_{i_m} = \bot$, then t is a leaf, therefore $\mathsf{next}^{\ell}_{C}(t)$ is the smallest sibling of t that is larger than itself. Hence, $\mathsf{next}^{\ell}_{C}(t) = \langle \omega_1 c_{i_1}, \ldots, \omega_{m-1} c_{m-1}, \mathsf{nextstring}^{a}(\omega_m) c \rangle$ where c is the minimum colour in $C^{\bot} \setminus \{c_{i_1}, \ldots, c_{i_{m-1}}\}$ and $a = \ell - \sum_{i=1}^{m-1} |\omega_i|$.

  Moreover, for $\omega_j = \lightning$, we say

$$\left\langle w_1 c_{i_1}, \ldots, w_{j-1} c_{i_{j-1}}, \lightning\, c_{i_j} \right\rangle = \begin{cases} \langle \omega_1 c_{i_1}, \ldots, \omega_{j-1} c \rangle & \text{if } c_{i_{j-1}} \neq \bot, \\ \left\langle \omega_1 c_{i_1}, \ldots, \mathsf{nextstring}^{a}(\omega_{j-1}) c' \right\rangle & \text{if } c_{i_{j-1}} = \bot, \end{cases}$$

Note that both of these tuples are $(j-1)-$tuples. Here, c is the smallest colour larger than $c_{i_{j-1}}$ in $C^{\bot} \setminus \{c_{i_1}, \ldots, c_{i_{j-2}}\}$, c' is the minimum color in $C \setminus \{c_{i_1}, \ldots, c_{j-2}\}$ and $a = \ell - \sum_{i=1}^{j-2} |w_i|$.

The value $\lightning$ is assigned to the last entry of $\mathsf{next}^{\ell}_{C}(t)$ by the application of rules presented above, only when t is the largest of its siblings. In this case, we reassign $\mathsf{next}^{\ell}_{C}(t)$ to the smallest sibling of t's parent that is larger than itself, as given above. Similarly, if $t = (\lightning \cdot \bot)$, then $t = \top$, since $\mathcal{L}^{\ell}_{C}$ is out of nodes.

We conclude this detailed computation of next^{ℓ}_{C} with the observation that the above computation takes only time $\mathcal{O}(k \log(k) \ell)$. □

Proposition 9.4.10. *Given a node t in the $\mathbb{W}$-labelled (c_0, C)-colourful tree $\mathcal{L}^{\ell}_{C}$, with at most 2^{ℓ} leaves and $K \subseteq C$ such that $\mathsf{colour}(t) \in K$, the next node larger than t such that $\mathsf{ColourSet}(t) \subseteq K \cup \{\bot\}$ can be found in time $\mathcal{O}(k \log(k) \ell)$, where $k = |C \cup c_0|$.*

Proof. For any node $t := \langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m} \rangle$ we know $c_{i_m} \in K$. We first find largest position s, such that $\mathsf{ColourSet}\left(\langle \omega_1 c_{i_1}, \ldots, \omega_s c_{i_s} \rangle\right) \subseteq K$.

We then compute the next node t' to $\langle \omega_1 c_{i_1}, \ldots, \omega_{s+1} c_{i_{s+1}} \rangle$ which also only has colours from the set K, that is $\mathsf{ColourSet}(t') \subseteq K$. But for the tree $\mathcal{L}^{\ell}_{C}$ constructed, consider the smallest colour c such that c is the smallest colour in $(K \cup \{\bot\}) \setminus$

$\{c_{i_1}, c_{i_2}, \ldots, c_{i_s}\}$ larger than $c_{i_{s+1}}$. Observe that the above set is non-empty as such colour exists, as $\bot \in (K \cup \{\bot\}) \setminus \{c_{i_1}, c_{i_2}, \ldots, c_{i_s}\}$. Hence we just need to return $\langle \omega_1 c_{i_1}, \ldots, \omega_s c_{i_s}, \omega_{s+1} c \rangle$, which in the tree constructed always exists and the smallest node larger than t such that $\mathsf{ColourSet}(t) \subseteq K \cup \{\bot\}$.

This indeed takes only time linear in the size of the encoding of a node, which is $\mathcal{O}(k \log(k) \ell)$. □

9.5 Strahler number of a Rabin game

In Chapter 4, we identified Strahler number as an important and natural parameter for parity games and was established to be equivalent to the register number also defined as a measure of parity games [LB20]. The Strahler number of a parity game is defined for each player and it captures the complexity of the cycles the opponent player can trap a player in [DJT20].

The contribution of this section is three-fold. Firstly, we show that the definition of the Strahler number of a Rabin game extends naturally from parity games. Secondly, we define and construct 'small' colourful Strahler universal trees, closely following the construction in Chapter 4. Finally, we provide a lifting algorithm on such colourful Strahler universal trees thus constructed.

Strahler number of a colourful tree. Similar to the Strahler number of a tree, we define the Strahler number of a (c_0, C)-colourful tree, as the Strahler number of the underlying tree within a colourful tree. More formally, we define the Strahler number of a colourful tree inductively and say that the Strahler number of any $(c_0, \varnothing)$-colourful tree is 1. For (c_0, C)-colourful trees $(c_0, \langle \mathcal{T}_1, \ldots, \mathcal{T}_m \rangle)$, it is defined as the maximum of $\mathrm{Str}(\mathcal{T}_i)$ among $\mathcal{T}_i$s rooted with a colour from C if the maximum value is obtained from a unique $\mathcal{T}_j$. Else, it is one more than the maximum of $\mathrm{Str}(\mathcal{T}_i)$ among the trees $\mathcal{T}_i$ for all i.

Tree of decomposition of a Rabin game. Recall the definition of a (c_0, C)-colourful decomposition of a Rabin game. We inductively define a tree of a given decomposition, which captures the shape of the decomposition.

For a $(c_0, \varnothing)$-colourful decomposition $\mathcal{D} = \langle A \rangle$, we define the tree $\mathcal{T}_\mathcal{D}$ as $\langle\rangle$ rooted at c_0. If on the other hand, we have a (c_0, C)-colourful decomposition $\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$, then we define the (c_0, C)-colourful tree as $\mathcal{T}_\mathcal{D} = \left(c_0, \left\langle \left(c_1, \mathcal{T}_{\mathcal{D}_1}\right), \ldots, \left(c_j, \mathcal{T}_{\mathcal{D}_j}\right) \right\rangle\right)$ where each $\mathcal{T}_{\mathcal{D}_i}$ is the recursively obtained colourful tree rooted with colour c_i.

Strahler number of a Rabin game. It is defined as the minimum of the Strahler numbers of all the trees of decompositions of the set of winning vertices for Steven in a Rabin game $\mathcal{G}$. Although Strahler number of a parity game is defined for both the players, for Rabin games we will stick to just Steven, since there is no natural notion of decomposition for Audrey in a Rabin game.

9.5.1 Colourful Strahler universal trees

Similar to Chapter 5 for ordered trees, we define a $\mathbb{W}$-labelled (c_0, C)-colourful tree $\mathcal{S}^{\ell,s}_{(c_0,C)}$ such that any (c_0, C)-colourful tree with Strahler number at most s, and at most 2^ℓ many leaves can be embedded into $\mathcal{S}^{\ell,s}_{(c_0,C)}$.

In the following sub-section, we again define the corresponding colourful version of a Strahler universal trees and prove their universality.

Strahler universality for colourful trees. A (c_0, C)-colourful tree is said to be s-Strahler n-universal if it can embed any (c_0, C)-colourful tree with n leaves, whose Strahler number is at most s. The tree we require is defined with the help of mutually inductive constructions of $\mathcal{S}^{\ell,s}_{(c_0,C)}$ and $\mathcal{W}^{\ell,s}_{(c_0,C)}$ as follows:

1. if $C = \emptyset$, and $s = 1$, then

$$\mathcal{S}^{\ell,s}_{(c_0,C)} = \left(c_0, \left\langle (\bot, \langle\rangle)^{2^\ell} \right\rangle\right)$$

2. if $C \neq \emptyset$ and $s = 1$ then

$$\mathcal{S}^{\ell,s}_{(c_0,C)} = \left(c_0, \left\langle (\bot, \langle\rangle)^{2^\ell}, \mathcal{S}^{\ell,s}_{(c_{i_1},C_1)}, \ldots, \mathcal{S}^{\ell,s}_{(c_{i_h},C_h)}, (\bot, \langle\rangle)^{2^\ell} \right\rangle\right),$$

   where $C_i = C \setminus \{c_i\}$ henceforth;

3. if $|C| \geqslant s - 1 > 0$ and $\ell = 0$ then

$$\mathcal{S}^{\ell,s}_{(c_0,C)} = \mathcal{W}^{\ell,s}_{(c_0,C)} = \left(c_0, \left\langle \mathcal{S}^{\ell,s-1}_{(c_{i_1},C_1)}, \ldots, \mathcal{S}^{\ell,s-1}_{(c_{i_h},C_h)}, (\bot, \langle\rangle) \right\rangle\right)$$

4. if $|C| \geqslant s - 1 > 0$ and $\ell \geqslant 1$ then

$$\mathcal{W}^{\ell,s}_{(c_0,C)} = \mathcal{W}^{\ell-1,s}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{S}^{\ell,s-1}_{(c_{i_1},C_1)}, \ldots, \mathcal{S}^{\ell,s-1}_{(c_{i_h},C_h)}, (\bot, \langle\rangle) \right\rangle\right) \cdot \mathcal{W}^{\ell-1,s}_{(c_0,C)}$$

5. if $|C| = s - 1 > 0$ and $\ell \geqslant 1$ then $\mathcal{S}^{\ell,s}_{(c_0,C)} = \mathcal{W}^{\ell,s}_{(c_0,C)}$

6. if $|C| > s \geqslant 2$ and $\ell \geqslant 1$ then

$$\mathcal{S}^{\ell,s}_{(c_0,C)} = \mathcal{W}^{\ell,s}_{(c_0,C)} \cdot \left(c_0, \left\langle \mathcal{S}^{\ell,s}_{(c_{i_1},C_1)}, \ldots, \mathcal{S}^{\ell,s}_{(c_{i_h},C_h)} \right\rangle\right) \cdot \mathcal{W}^{\ell,s}_{(c_0,C)}$$

Proposition 9.5.1. *The constructed (c_0, C)-colourful tree is $\mathcal{S}^{\ell,s}_{(c_0,C)}$ is s-Strahler 2^ℓ-universal.*

Proof. For $n \leqslant 2^\ell$, for a (c_0, C)-colourful tree $\mathcal{T}$, which has n leaves, we show $\mathcal{T}$ can be embedded in

1. $\mathcal{W}^{\ell,s}_{(c_0,C)}$ if each $\mathcal{T}_i$ such that $\mathcal{T} = \left(c_0, \left\langle \mathcal{T}_1, \ldots, \mathcal{T}_j \right\rangle\right)$ has Strahler number at most $s-1$;
2. $\mathcal{S}^{\ell,s}_{(c_0,C)}$ if $\mathcal{T}$ has Strahler number at most s.

The details of the proof are omitted as it is very close to the proof of Lemma 5.1.2 in Chapter 4.

If $C = \varnothing$, and for any ℓ, the Strahler number of $\mathcal{T}$ is 1, and therefore the tree can always be embedded in $\left\langle (\bot, \langle\rangle)^n \right\rangle$, and the above statement is true.

Similarly, for any tree with $s = 1$, since the Strahler number is 1, each node in the tree must have at most one child not coloured with $\bot$. Therefore, a tree $\mathcal{T}$ with n leaves and Strahler number 1 is of the form $\left\langle (\bot, \langle\rangle)^{n_1}, \mathcal{T}_i, (\bot, \langle\rangle)^{n_2} \right\rangle$, where $n_1 + n_2 \leqslant n$, and $\mathcal{T}_i$ is a $(c_i, C \setminus \{c_i\})$ colourful with Strahler number 1.

For any tree with $\ell = 0$, its Strahler number can be exactly 1, and again, any tree $\mathcal{T}$ would have at most one child. This tree can therefore be expressed as $(c_i, \langle \mathcal{T}_i \rangle)$. If the child is $\langle (\bot, \langle\rangle) \rangle$, then trivially this is embedded. If the child is coloured with $c_i \neq \bot$, then $\mathcal{S}^{\ell,s-1}_{(c_i,C_i)}$ embeds this tree. Therefore, so do both $\mathcal{S}^{\ell,s}_{(c_0,C)}$ and $\mathcal{W}^{\ell,s}_{(c_0,C)}$.

Now we move to the case where $\mathcal{T} = \left(c_0, \left\langle \mathcal{T}_1, \ldots, \mathcal{T}_j \right\rangle\right)$ has strictly more than 1 leaf, and Strahler number at least 2. So we have $\ell > 0$, $s \geqslant 2$ and $|C| \geqslant 1$.

We show that if the Strahler number of each children of $\mathcal{T}$ is at most $s-1$, then $\mathcal{T}$ can be embedded in $\mathcal{W}^{\ell,s}_{(c_0,C)}$ as claimed. This is because one can find a value $p \in [1, j]$ such that $\mathcal{T}$ can be expressed as $\mathcal{T}_{\text{left}} \cdot (c_0, \langle \mathcal{T}_p \rangle) \cdot \mathcal{T}_{\text{right}}$ where both $\mathcal{T}_{\text{left}}$ and $\mathcal{T}_{\text{right}}$ have at most $n/2$ leaves, and $\mathcal{T}_p$ has Strahler number at most $s-1$. Therefore, each $\mathcal{T}_{\text{left}}$ and $\mathcal{T}_{\text{right}}$ can be embedded in $\mathcal{W}^{\ell-1,s}_{(c_0,C)}$ since $n/2 \leqslant 2^{\ell-1}$. Now $\mathcal{T}_p$ has Strahler number at most $s-1$ and is $C \setminus \{c_{i_p}\}$-colourful. Therefore, $\mathcal{T}_p$ can inductively be embedded into $\mathcal{S}^{\ell,s}_{(c_{i_p},C_p)}$ where $C_p = C \setminus \{c_{i_p}\}$.

If $|C| = s - 1$, indeed all children $\mathcal{T}_i$ have Strahler number at most $s-1$, and therefore can be embedded in $\mathcal{W}^{\ell,s}_{(c_0,C)}$ by induction and therefore also in $\mathcal{S}^{\ell,s}_{(c_0,C)}$, which is defined to be identical.

If not and $|C| > s-1$, then (at most) one of its children, say $\mathcal{T}_p$ can have Strahler number s. Suppose $\mathcal{T} = \mathcal{T}_{\text{left}} \cdot \big(c_0, \langle \mathcal{T}_p \rangle\big) \cdot \mathcal{T}_{\text{right}}$, where $\mathcal{T}_p$ is a $(c_{i_p}, C \setminus \{c_{i_p}\})$-colourful tree. Since none of the children of $\mathcal{T}_{\text{left}}$ or $\mathcal{T}_{\text{right}}$ would have children of Strahler number s, they can be embedded in $\mathcal{W}^{\ell,s}_{(c_0,C)}$ and since $\mathcal{T}_p$ does not use colour c_{i_p}, it can be embedded in $\mathcal{S}^{\ell,s}_{(c_{i_p},C_p)}$ where $C_p = C \setminus \{c_{i_p}\}$. □

We proceed to show that these trees constructed have size $(\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{k-2}{s-1}k!$.

Proposition 9.5.2. *If $|C| + 1 \geqslant s$, then the following inequality holds for $S(k-1, s, \ell)$, the number of leaves of $\mathcal{S}^{\ell,s}_{(c_0,C)}$, where $|C| = h = k-1$,*

$$S(k-1,s,\ell) \leqslant (\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{k-2}{s-1}k!.$$

Proof. We prove this by mutual induction on the following two inequalities.

$$W(h,s,\ell) \leqslant (\ell+2)2^{\ell+s-1}\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}(h+1)!$$

$$S(h,s,\ell) \leqslant (\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{h-1}{s-1}(h+1)!$$

where $W(h,s,\ell)$ denotes the number of leaves of $\mathcal{W}^{\ell,s}_{(c_0,C)}$ when $\ell > 0$, $|C|+1 \geqslant s > 1$. From the definition of the trees, we now inductively see how the terms $S(h,s,\ell)$ and $W(h,s,\ell)$ are defined:

1. if $h = 0$, $s = 1$, and any $\ell > 0$ then $S(0,1,\ell) = 2^\ell$
2. if $h \neq 0$ and $s = 1$ then $S(h,1,\ell) = hS(h-1,1,\ell) + 2^{\ell+1}$.
3. if $h \geqslant s-1 > 0$ and $\ell = 0$ then $S(h,s,\ell) = W(h,s,\ell) = hS(h-1,s-1,\ell) + 1$
4. if $h \geqslant s-1 > 0$ and $\ell \geqslant 1$ then $W(h,s,\ell) = 2W(h,s,\ell-1) + hS(h-1,s-1,\ell) + 1$
5. if $h = s-1 > 0$ and $\ell \geqslant 1$ then $S(h,s,\ell) = W(h,s,\ell)$
6. if $h > s \geqslant 2$ and $\ell \geqslant 1$ then $S(h,s,\ell) = 2W(h,s,\ell) + hS(h-1,s,\ell)$

For $s = 1$ and $h = 0$. $S(0,1,\ell) = 2^\ell$ from Item 1.

For $s = 1$, and $h > 0$. we will show that $S(h, 1, \ell) \leqslant 2^{\ell+1}(h+1)!$. The proof follows from direct induction. In the base case we have $h = 1$ from Item 1.

Since $S(h, 1, \ell) = h \cdot S(h-1, 1, \ell) + 2^{\ell+1} \leqslant h \cdot 2^{\ell+1} h! + 2^{\ell+1} = (h)2^{\ell+1} h! + 2^{\ell+1} \leqslant 2^{\ell+1} \cdot (h+1)!$.

Consider $h > s - 1 > 0$ and $\ell = 0$. then we get $S(h, s, \ell) = W(h, s, \ell) \leqslant 2 \cdot h \cdot h!$. Once more, we proceed by induction. We already know that for $h = s = 1$, $S(1, 1, \ell) \leqslant 2^2$ For values of h larger than 1, we have $S(h, s, \ell) = W(h, s, \ell) = h \cdot S(h-1, s-1, \ell) + 1 \leqslant h \cdot 2 \cdot (h-1)!(h-1) + 1 \leqslant 2 \cdot h \cdot h!$

Consider $h \geqslant 1$, $s = 2$, and $\ell \geqslant 1$. We will show by induction that for $s = 2$, $W(h, s, \ell) \leqslant (\ell+2)2^{\ell+s-1} \cdot (h+1)!$.

For $h = 1$, we show $W(1, 2, \ell) \leqslant (\ell+2)2^{\ell+1} - 1$.

We know by the previous case where $\ell = 0$, $W(1, 2, 0) \leqslant 2 \leqslant (\ell+2)2^{\ell+1} - 1$. We assume the claim holds for $\ell - 1$ and proceed by induction,

$$\begin{aligned} W(1, 2, \ell) &= 2W(1, 2, \ell-1) + S(0, 1, \ell) + 1 \\ &\leqslant 2\left((\ell+1)2^{\ell} - 1\right) + \left(2^{\ell}\right) + 1 \\ &\leqslant (\ell+1)2^{\ell+1} - 2 + 2^{\ell} + 1 \\ &= (\ell+2)2^{\ell+1} - 1 \end{aligned}$$

For $h \geqslant 2$ and $s = 2$, we will show $W(h, s, \ell) \leqslant (\ell+2)2^{\ell+s-1} \cdot (h+1)h!$

$$\begin{aligned} W(h, 2, \ell) &= 2W(h, 2, \ell-1) + hS(h-1, 1, \ell) + 1 \\ &\leqslant 2\left(((\ell-1)+2)2^{\ell+s-2}(h+1)!\right) + hS(h-1, 1, \ell) + 1 \\ &\leqslant 2\left((\ell+1)2^{\ell} \cdot (h+1)!\right) + h\left(2^{\ell+1} h!\right) + 1 \\ &\leqslant (\ell+1)2^{\ell+1} \cdot (h+1)! + 2^{\ell+1} \cdot h \cdot h! + 1 \\ &\leqslant (\ell+2)2^{\ell+1} \cdot (h+1)! \end{aligned}$$

In the above, we use the inequality $S(h, 1, \ell) \leqslant 2^{\ell+1}(h+1)!$ from the case where $s = 1$.

Consider $h > s-1 \geqslant 2$ and $\ell \geqslant 1$. We will show the bound $W(h,s,\ell) \leqslant (\ell+2)2^{\ell+s-1}\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}\cdot(h+1)!$ on W inductively

$$\begin{aligned}
W(h,s,\ell) &= 2W(h,s,\ell-1) + hS(h-1,s-1,\ell) + 1 \\
&\leqslant 2\left((\ell+1)2^{\ell+s-2}\binom{\ell+s-3}{s-2}\binom{h-2}{s-2}(h+1)!\right) \\
&\qquad + h\left((\ell+2)2^{\ell+s-1}\binom{\ell+s-3}{s-3}\binom{h-2}{s-2}h!\right) + 1 \\
&\leqslant (\ell+2)\left(2^{\ell+s-1}\left\{\binom{\ell+s-3}{s-2} + \binom{\ell+s-3}{s-3}\right\}\binom{h-2}{s-2}(h+1)!\right) \\
&\leqslant (\ell+2)2^{\ell+s-1}\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}(h+1)!
\end{aligned}$$

Consider $h = s-1 > 0$ and $\ell \geqslant 1$. In this case, we have

$$\begin{aligned}
S(h,s,\ell) = W(h,s,\ell) &\leqslant (\ell+2)2^{\ell+s-1}\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}(h+1)! \\
&\leqslant (\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{h-1}{s-1}(h+1)!
\end{aligned}$$

Consider $h > s \geqslant 2$ and $\ell \geqslant 1$. Suppose for all $j < h$ and $s' < s$, we have

$$S(j,s',\ell) \leqslant 2^{\ell+s'}(\ell+2)\binom{\ell+s'-2}{s'-2}\binom{j-1}{s'-1}(j+1)!$$

$$\begin{aligned}
&S(h,s,\ell) = 2W(h,s,\ell) + hS(h-1,s,\ell) \\
&\leqslant 2\left((\ell+2)2^{\ell+s-1}\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}(h+1)!\right) + h\left((\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{h-2}{s-1}h!\right) \\
&= 2^{\ell+s}(\ell+2)\binom{\ell+s-2}{s-2}\binom{h-2}{s-2}(h+1)! + 2^{\ell+s}(\ell+2)\binom{\ell+s-2}{s-2}\binom{h-2}{s-1}hh! \\
&\leqslant (\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\left\{\binom{h-2}{s-2} + \binom{h-2}{s-1}\right\}(h+1)! \\
&\leqslant (\ell+2)2^{\ell+s}\binom{\ell+s-2}{s-2}\binom{h-1}{s-1}(h+1)!
\end{aligned}$$

Our claim about the bounds of $S(k-1,s,\ell)$ follow from the fact that $k = h-1$. $\square$

9.5.2 Lifting Algorithm Using Colourful Strahler Universal Trees

Here we provide a bit-string labelling for Strahler universal Trees $\mathcal{S}^{\ell,s}_{(c_0,C)}$ such that any (c_0, C)-colourful tree with Strahler number at most s, and at most 2^ℓ many leaves can be embedded into $\mathcal{S}^{\ell,s}_{(c_0,C)}$.

For the last time, we revisit the bit-string order on $\mathbb{W} = \{0,1\}^*$, and use vocabulary from Chapter 5 that we recall here. A word $\omega \in \mathbb{W}$, if $\omega = 0 \cdot \omega'$ then 0 is a leading bit, and if $\omega = 1 \cdot \omega'$ then 1 is a leading bit of ω. All the bits in ω' defined above would be non-leading bits.

We define labelled Strahler universal tree below again to help navigate these recursively defined trees efficiently. These labelled colourful trees $\mathcal{J}^{\ell,s}_C$ and $\mathcal{K}^{\ell,s}_C$ are the $\mathbb{W}$-labellings of the colourful Strahler universal trees $\mathcal{S}^{\ell,s}_{(c_0,C)}$ and $\mathcal{W}^{\ell,s}_{(c_0,C)}$, respectively.

1. if $C = \varnothing$, and $s = 1$, then

$$\mathcal{J}^{\ell,s}_C = \left\{ \left(0^{\ell-1}\bot, \ldots, 1^{\ell-1}\bot \right) \right\}$$

2. if $C \neq \varnothing$ and $s = 1$ then

$$\mathcal{J}^{\ell,s}_C = \left[\mathcal{J}^{\ell,s}_\varnothing \right]^0 \bigcup_i \left(\epsilon c_i \odot \mathcal{J}^{\ell,s}_{C_i} \right) \cup \left[\mathcal{J}^{\ell,s}_\varnothing \right]^1$$

   where $C_i = C \setminus \{c_i\}$;

3. if $|C| \geqslant s - 1 > 0$ and $\ell = 0$ then

$$\mathcal{K}^{\ell,s}_C = \bigcup_i \left(\epsilon c_i \odot \mathcal{J}^{\ell,s-1}_{C_i} \right) \cup \langle \epsilon\bot \rangle$$

   and

$$\mathcal{J}^{\ell,s}_C = \left[\mathcal{K}^{\ell,s}_C \right]^0 = \bigcup_i \left(0c_i \odot \mathcal{J}^{\ell,s-1}_{C_i} \right) \cup \langle 0\bot \rangle$$

4. if $|C| \geqslant s - 1 > 0$ and $\ell \geqslant 1$ then

$$\mathcal{K}^{\ell,s}_C = \left[\mathcal{K}^{\ell-1,s}_C \right]^0 \bigcup_i \left(\epsilon c_i \odot \mathcal{J}^{\ell,s-1}_{C_i} \right) \cup \left[\mathcal{K}^{\ell-1,s}_C \right]^1$$

5. if $|C| = s - 1 > 0$ and $\ell \geqslant 1$ then $\mathcal{J}^{\ell,s}_C = \mathcal{K}^{\ell,s}_C$

6. if $|C| > s \geqslant 2$ and $\ell \geqslant 1$ then

$$\mathcal{J}_C^{\ell,s} = \left[\, \mathcal{K}_C^{\ell,s} \,\right]^0 \bigcup_i \left(\epsilon c_i \odot \mathcal{J}_{C_i}^{\ell,s}\right) \cup \left[\, \mathcal{K}_C^{\ell,s} \,\right]^1$$

We argue that an operator similar to the one used to navigate the above colourful universal trees can be modified here to navigate this constructed Strahler universal tree.

Lemma 9.5.3. *The time taken to find the next node in the colourful tree $\mathcal{S}_C^{\ell,s}$ is bounded by $O\left((s+\ell)k \log k\right)$.*

Proof. Due to the similarity to the construction in Lemma 5.2.3, we get the following characterisation of the set of all nodes that are not coloured with a $\perp$: all prefixes of a k-length tuple $\left(\omega_1 c_{i_1}, \ldots, \omega_m c_{i_m}, \ldots, \omega_k c_{i_h}\right)$ form nodes in the tree if:

0. $c_{i_a} \neq c_{i_b}$ if $a \neq b \leqslant h$;
1. the number of bits used in all of $w_1, \ldots, w_h$ is at most $(s-1)+\ell$;
2. the number of non-empty ω_is is exactly $s-1$;

for each $j = 1, \ldots, h \leqslant |C|$, in this tuple,

3. if there are less than $s-1$ non-empty bit strings among $\omega_1, \ldots, \omega_j$, but there are ℓ non-leading bits used in them, then $\omega_{j+1} = 0$;
4. if all strings $\omega_j, \ldots, \omega_h$ are non-empty, then each of them has 0 as its leading bit.

Other than this, if there is a ancestor of the node $\left\langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m} \right\rangle$ of the above, then so is the sequence $\left\langle \omega_1 c_{i_1}, \ldots, \omega_m \perp \right\rangle$ a leaf in the tree. Moreover, the element $\left\langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m}, \omega'_{m+1} \perp \right\rangle$ is also a leaf, where $\omega' \in \mathbb{W}$ and $|\omega'_{m+1}| \leqslant \ell - b$, where b denotes the number of leading bits used in $\omega_1 \ldots, \omega_m$.

This gives us a characterisation of Strahler universal trees that allow for easy navigation, and a succinct encoding of nodes in the colourful tree. Using this, we argue that with only a small polynomial factor to the size of the trees constructed, we can obtain an effective lifting algorithm for games of bounded Strahler number.

We also modify the subroutine of computing the next sibling at a given level in the Strahler universal trees to also compute the next sibling in our construction of colourful Strahler universal trees. This in turn helps us compute the next node of the tree. Observe that for a node $\omega = \left\langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m} \right\rangle$, such that $c_{i_m} \neq \perp$, the tuple of bitstrings $\langle \omega_1, \ldots, \omega_m \rangle$ corresponds exactly to a node constructed in a labelled Strahler universal tree in Chapter 4.

- If $c_{i_m} \neq \perp$, then the SNext(ω) is the first child of ω and is therefore the node $\langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m}, \omega_{m+1} c \rangle$, where c is the minimum color in $C^{\perp} \setminus \{c_{i_1}, \ldots, c_{i_m}\}$.
  - if $c \neq \perp$, then $\omega_{m+1} = 0^t$ where t is obtained by finding the child of the node in a Strahler universal tree. Intuitively, t is the number of 0s to make the number of non-empty bit strings in some potential node equal to $s-1$, and the number of bits in such a node equal to $(s-1)+\ell$. More rigorously, t is obtained by computing:
    * the number bits left, denoted by b as $(s-1)+\ell-\sum_{i=1}^{m}|\omega_i|$;
    * the number of positions after m that can host ϵ, denoted by $e = (h-(s-1))-$number of ω_is which are already ϵ;
    * the number of positions after $m+1$ that cannot host an epsilon, denoted by $f = h-m-e-1$;

    and finally, we define $t = b - f$.
  - alternatively, if $c = \perp$, then $\omega_{m+1} = 0^t$ where $t = \ell - b$ where b is the number of leading bits among $\omega_1, \ldots, \omega_m$.
- If $c_{i_m} = \perp$, then the node w is a leaf, and we need to find a sibling of ω or the largest ancestor of w with a sibling. First, we check if there is a sibling of the given node. This is done by finding the value $t = \ell - b$, where b is the number of leading bits among $\omega_1, \ldots, \omega_m$. Later, if $\mathsf{nextstring}^t(\omega_m)$ is defined, then we declare SNext(ω) = $\langle \omega_1 c_{i_1}, \ldots, \mathsf{nextstring}^t(\omega_m) c_{i_m} \rangle$.

  If such a node is $\lightning$, then we compute its next value as follows. Let p be the largest position where the tuple $\langle \omega_1, \ldots, \omega_m \rangle$ has a next sibling when viewed as a node in the Strahler universal tree. Such a p is simply found by finding the largest $p \leqslant m$ such that at least one of the following is not true.

  1. the number of non-empty ω_is is $s-1$;
  2. the number of bits used in all of $\omega_1, \ldots, \omega_p$ is $(s-1)+\ell$;
  3. $\omega_p = 1^j$ for some $j > 0$ but the number of non-leading bits used in them is ℓ. The number of non-empty bit strings among $\omega_1, \ldots, \omega_p$ is ℓ.
  4. if $\omega_p = 01^j$ for some $j \geqslant 0$, the number of non-leading bits used in $\omega_1, \ldots, \omega_p$ is ℓ, and all bit strings $\omega_r, \ldots, \omega_m$ are non-empty.

  We also simultaneously find the largest position $q \geqslant m$ such that a following c exists where c is the smallest colour larger than c_q in $C^{\perp} \setminus \{c_{i_1}, \ldots, c_{i_q}\}$. If $q \geqslant p$, then we say SNext(ω) = $\langle \omega_1 c_{i_1}, \ldots, \omega_q c \rangle$. If $p > q$, then we find the

next node in the underlying Strahler universal tree at level p. Let this node be represented by the tuple $\langle \omega_1, \omega_2 \ldots, \omega_{p-1}, \omega'_p \rangle$, where ω'_p is obtained using the following two cases:

- If less than ℓ non-leading bits are used in $\omega_1, \ldots, \omega_p$, then we set ω'_p to be $\omega_p 10^b$ so that exactly t non-leading bits are used in $\omega_1, \ldots, \omega_{p-1}, \omega'_p$
- If exactly ℓ non-leading bits are used instead, then we let ω'_p to instead be defined such that $\omega_p = \omega'_p 01^j$.

We now return the node $\langle \omega_1 c_{i_1}, \ldots, \omega_{p-1} c_{i_{p-1}}, \omega'_p c \rangle$ where c is the smallest colour larger among $C^\perp \setminus \{c_{i_1}, \ldots, c_{i_{p-1}}\}$. □

Proposition 9.5.4. *Given a node in the $\mathbb{W}$-labelled (c_0, C)-colourful tree $\mathcal{J}_C^{\ell,s}$ and a subset of colours $K \subseteq C$, such that the colouring of the node is in K, the next node larger than it, whose set of colours used is contained in $K \cup \{\perp\}$ can be found in time $\mathcal{O}((s+\ell)k\log(k))$, where $k = |C \cup c_0|$.*

Proof. The proof is similar to that of Lemma 9.4.9. We, however, give it for completeness. For any node $t = \langle \omega_1 c_{i_1}, \ldots, \omega_m c_{i_m} \rangle$ we know $c_{i_m} \in K$. We first find largest position p such that $\mathsf{ColourSet}\left(\langle \omega_1 c_{i_1}, \ldots, x_s c_{i_p} \rangle\right) \subseteq K^\perp$, , where $K^\perp$ denotes $K \cup \{\perp\}$. Observe that this implies $c_{i_{p+1}} \in K$ and $c_{i_{p+1}} \neq \perp$.

We then compute the next node to $\langle \omega_1 c_{i_1}, \ldots, \omega_{p+1} c_{i_{p+1}} \rangle$ instead such that $\mathsf{ColourSet}(t)$ does not intersect with the set of colours $C \setminus K$. But for the tree $\mathcal{J}_C^{s,\ell}$ constructed, consider the smallest colour $c \in K^\perp \setminus \{c_{i_1}, c_{i_2}, \ldots, c_{i_p}\}$ such that c is also larger than $c_{i_{p+1}}$. The above set is non-empty since it contains $\perp$. Hence we just need to return $\langle \omega_1 c_{i_1}, \ldots, \omega_s c_{i_s}, \omega_{s+1} c \rangle$ which is the smallest node larger than t such that $\mathsf{ColourSet}(t)$ is contained in $K^\perp$. This indeed takes only time linear in the size of the encoding of a node, which is $\mathcal{O}((s+\ell)k\log(k))$. □

From Theorem 9.3.4, we know that our lifting algorithm can be performed on any tree and from Theorem 9.5.5, we know that small Strahler universal trees exist and we know from Lemma 9.5.3 and Proposition 9.5.4 that these trees can be navigated effectively. Combining these results, we get the following theorem.

Theorem 9.5.5. *A (c_0, C)-colourful Rabin game with n vertices m edges, of Strahler number s, and $k = |C| + 1$, where $s \leqslant \min\{\lceil \lg n \rceil, k\}$ can be solved in space that is $\mathcal{O}(nk \log n \lg k)$ and time*

$$\widetilde{\mathcal{O}}\left(nmk2^s \binom{\lceil \lg n \rceil + s - 2}{s-2} \binom{k-2}{s-1} k!\right).$$

Chapter 10

Rabin games against a fair opponent

A motivation to work on Rabin games is to solve reactive synthesis of systems based on given high-level specifications. However in many cases, solutions to the synthesis problem do not exist for mundane reasons. For instance, consider the case where a machine can be synthesised as long as the input sequence is restricted to a specific language. Alternatively, in some other case, solutions could exist if some rationality is assumed on behalf of the environment the synthesised machine interacts with. Indeed, the former setting where the input was restricted to LTL formulas was considered by Chatterjee, Henzinger and Jobstmann [CHJ08] and the latter model was considered in the works of Fisman, Kupferman, and Lustig [FKL10] and also Kupferman, Perelli, and Vardi [KPV16].

One such case of inability to solve the synthesis problem could be due to the presence of "unfair" executions of a synthesised model. In such cases, solutions to the synthesis problem could benefit if additional fairness constraints are imposed on its executions. In fact, with a similar motivation, this problem was recently studied in the work of Banerjee et al. [BMM$^+$22] where they considered Rabin games with an additional condition of *strong transition* fairness [QS83] for the environment. Their contribution was a symbolic algorithm for the above problem that took $\mathcal{O}(n^{k+1}(k)!)$ symbolic steps. However, they left open the question of finding effective enumerative algorithms.

In this chapter, we consider Rabin games with strong transition fairness and provide algorithms that match the running time of algorithms to solve Rabin games (without any additional fairness conditions). We generalise our algorithms from Chapter 9 to also identify the winner in Rabin games with *transition fairness* [QS83,

BMM+22], which therefore also identifies a winner of almost-sure winning conditions of turn-based stochastic Rabin games [CdAH05].

In a game with transition fairness, a set of edges starting from environment vertices are marked "live". If the game visits a source vertex of a live edge infinitely often, Audrey must ensure that this live edge is also traversed infinitely often. In a stochastic game, instead of a partition of the vertex set among Steven and Audrey, we have a tri-partition of the set of vertices between Steven, Audrey and the remaining vertices marked "random." When the game visits a random vertex, one of its neighbours is chosen uniformly at random. Almost-sure winning conditions for stochastic games are a special case of games with transition fairness: allocate all random vertices to the environment and mark every outgoing edge from a random vertex to be live.

We first show that the winning region of a fair Rabin game has a *colourful fair decomposition*. Since colourful fair decompositions are also naturally associated to colourful trees, we ask if one can indeed construct an algorithm that solves these games using our universal colourful trees defined in the Chapter 9. We answer this question positively and provide a simple progress measure lifting algorithm to solve such games. Together with our universal colourful trees defined in the Chapter 9, we obtain a $\widetilde{\mathcal{O}}\big(mn^2(k!)^{1+o(1)}\big)$-time and $\mathcal{O}(nk \lg k \lg n)$-space algorithm for Rabin games with transition fairness as well as for almost sure winning in turn-based stochastic Rabin games. While there is a known reduction from Rabin games with transition fairness to usual Rabin games with at most nk vertices, our algorithm shaves off a k^3 factor from the above in the worst case.

10.1 Games with live edges

Consider a Rabin game in which a subset L of the set of Audrey's edges (out-going from Audrey's vertices) are identified as *live edges*. A play in this game is *fair* with respect to this set L of live edges if for every live edge $u \to v$, if u is visited infinitely often, then the edge $u \to v$ is taken infinitely often. Alternately, one can state that for each edge, if this path visits the source of any edge in L infinitely often, then it also visits the target of that edge infinitely often. An infinite path satisfies the *fair Rabin condition* with respect to L if it is not fair with respect to L or satisfies the Rabin condition.

A (c_0, C)-colourful *fair* Rabin game $\mathcal{G}$ is defined similarly to Rabin games and consists of an underlying (c_0, C)-colourful Rabin game whose vertices are partitioned V_A and V_S, belonging to Audrey and Steven, respectively, and a subset $L \subseteq E$ of

live edges such that the source vertex of each edge in L is owned by Audrey. A fair Rabin game is said to be winning from a vertex for Steven if there is a positional strategy $\sigma \subseteq E$ of Steven such that all infinite paths starting from this vertex in the graph obtained restricted to the set of edges σ satisfy the fair Rabin condition. One can, in polynomial time, convert a fair Rabin game to a Rabin game with a similar set of vertices and edges (live edges become normal edges) where the winners are preserved. Although this reduction shows that these fair Rabin games have a positional winning strategy for Steven, it increases the number of colours in the new Rabin game.

Proposition 10.1.1. *For a (c_0, C)-colourful Rabin game, with live edges L, there is a Rabin measure into an $\mathbb{L}$-labelled (c_0, C_L)-colourful tree, where the set $C_L = C \cup \{c_e \mid e \in L\}$, where all infinite paths satisfy the Rabin condition.*

Proof. Observe that a (c_0, C)-colourful graph $\mathcal{G}$ with Rabin, with live edges L, can be encoded as a (c_0, C_L)-colourful Rabin game over the same graph where the set of colours $C_L = C \cup \{c_e \mid e \in L\}$. For any live edge $e = u \to v$, we redefine the bad sets and good sets of colours from $\mathcal{G}$, that is, B_v and G_u to include c_e in addition to the other colours assigned to it by the game $\mathcal{G}$ with live edges. □

We list two ways to find if Steven wins from a specific vertex using known techniques.

Approach one: The above Rabin condition on a (c_0, C)-colourful game with strong transition fairness with live edges enlisted in a set L can be converted into a $(c_0, C \cup L)$-colourful Rabin game with no fairness conditions imposed on Audrey.

On a Rabin game with n vertices, k colours and t live edges, we have an algorithm that runs in time $\widetilde{\mathcal{O}}\big(mn^2(k+t+1)!^{1+o(1)}\big)$ time.

Approach two: One could use gadgets constructed in the work of Chatterjee, de Alfaro and Henzinger [CdAH05] to show almost-sure winning condition for Rabin games can be reduced to solving Rabin games without stochastic vertices. A direct modification of their gadget would give us a method to convert a (c_0, C)-colourful Rabin game with live edges L into a (c_0, C)-colourful Rabin game without fairness and with at most $\mathcal{O}(tk+n)$ many vertices. This takes $\widetilde{\mathcal{O}}\big((m+tk)(n+tk)^2(k)!^{1+o(1)}\big)$ time from Algorithm 10 from Theorem G. This reduces the exponential dependence on $(k+t)!$ already to a $k!$. We do not elaborate on these gadgets, but refer a reader to the work of Chatterjee, de Alfaro and Henzinger [CdAH05] or to the work of

Chatterjee, Henzinger and Jurdziński [CHJ05] where similar gadgets are used to find if Steven wins almost surely in stochastic Rabin and parity games, respectively.

Our main contribution is our characterisation of Rabin games with fairness using *colourful fair decomposition*. This leads to a lifting algorithm for fair Rabin games that is faster by a polynomial factor than the second approach.

10.2 Colourful fair decomposition

Following a pattern similar to Chapter 9, here we define a colourful fair decomposition and a measure for fair Rabin games, and show that these exactly characterise games where Steven wins.

Colourful fair decomposition. Consider a (c_0, C)-colourful Rabin game $\mathcal{G}$ with the arena (V, E), sets of good colours $\{G_v\}$ and bad colours $\{B_v\}$ for each vertex $v \in V$ and a set of *live edges* L. For a fixed positional strategy σ of Steven, a (c_0, C)-colourful fair decomposition $\mathcal{D}$ is defined on the game $\mathcal{G}|_\sigma$ obtained by restricting to the strategy edges. For a fixed σ, the decomposition $\mathcal{D}$ of $\mathcal{G}|_\sigma$ is a recursive subdivision of vertices of $\mathcal{G}$ into subsets of vertices that satisfies specific conditions. If $C = \emptyset$, then we say $\mathcal{D} = \langle V \rangle$ is a (c_0, C)-colourful fair decomposition all fair paths in $\mathcal{G}|_\sigma$ if all fair paths from all vertices in V visit a vertex v such that $c_0 \in G_v$. Else, if C and V are non empty, we say that the decomposition

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

satisfies the following conditions if there is a fixed positional Steven strategy such that:

1. A is the set of all vertices in V such that all fair paths starting from A in $\mathcal{G}|_\sigma$ visit some vertex $v \in V$ such that $c_0 \in G_v$;

and setting $\mathcal{G}_1 = V \setminus A$. For $i \in \{1, \ldots, j\}$, we have

2. V_i is a set of vertices such that there are no fair paths in $\mathcal{G}_i|_\sigma$ which start at a vertex in V_i and visit a vertex in $\mathcal{G}_i \setminus V_i$ and $c_i \notin B_v$ for all $v \in V_i$;

3. $\mathcal{D}_i$ is a $(c_i, C \setminus \{c_i\})$-colourful fair decomposition of V_i (with the same Steven strategy σ);

4. A_i is the set of all vertices in $\mathcal{G}_i|_\sigma$ such that all fair paths from A_i within $\mathcal{G}_i$ visits some vertex in V_i;

5. $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$;

and we have $\mathcal{G}_{j+1} = \varnothing$.

Instead of producing a definition with attractors in games, as we have done for attractor decompositions as well as for colourful decompositions, we only deal with games obtained on restriction to a strategy where all fair paths satisfy the Rabin condition. However, if one wishes so, one could instead consider the nearly identical definitions of colourful decompositions, only to replace Steven attractors with a modified definition of "fair attractors" defined for Steven, and "fair traps" defined for Audrey. But to avoid many new concepts, we restrict ourselves to paths in strategy graphs.

A measure for live Rabin games. Consider a map λ from the vertices of a (c_0, C)-colourful game $\mathcal{G}$ to an $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$ which contains an additional $\top$ element. An edge $u \to v$ is said to be *live consistent* in a mapping λ if it satisfies the condition G_ℓ defined below and B, defined in Section 9.2.

(G_ℓ) $\mathsf{colour}(\lambda(u)) = \bot$ and moreover, $\mathsf{GCA}(\lambda(u), \lambda(v)) = \mathrm{parent}(\lambda(u))$

This intuitively says that the measure along this edge might potentially increase, but not larger that the last descendent of its parent. We say a vertex v is *live-consistent* if

- it has *at least one* edge that is satisfying ($\mathsf{G}_\succ$ AND B), and
- every other outgoing edge satisfies (G_ℓ AND B).

For every fair infinite run that visits this live-consistent vertex infinitely often also satisfies $\mathsf{G}_\succ$ infinitely often. This is because along the live edge that is taken for this run to be fair, the measure along this live edge decreases. Finally, we say a mapping λ is *live consistent* if all vertices are consistent or live consistent.

Theorem 10.2.1. *Given a (c_0, C)-colourful Rabin game $\mathcal{G}$ and a designated set of live edges $L \subseteq E$ then the following statements are equivalent:*

1. *Steven has a positional strategy σ such that all infinite paths in the restricted game $\mathcal{G}|_\sigma$ satisfy the fair-Rabin condition.*
2. *there is a (c_0, C)-colourful fair decomposition of vertices of V.*
3. *there is a live-consistent map λ from vertices of $\mathcal{G}$ to a $\mathbb{L}$-labelled (c_0, C)-colourful tree rooted at colour c_0;*

Lemma 10.2.2 ((1 $\implies$ 2) of Theorem 10.2.1)**.** *Given an (c_0, C)-colourful Rabin graph and a set L of live edges. If all paths satisfy the Rabin condition, then there is a (c_0, C)-colourful fair decomposition.*

Proof. We construct such a fair colourful decomposition by fixing the same strategy σ and constructing a decomposition inductively on the sum of the number of colours in C and the number of vertices in $\mathcal{G}$.

Base case. If $C = \varnothing$, then for all vertices v, $B_v = \varnothing$. All fair paths in the SCCs after finitely many steps must visit a vertex v such that $G_v = \{c_0\}$. This is because all fair paths in $\mathcal{G}|_\sigma$ satisfy the Rabin condition. The (c_0, C)-colourful fair decomposition is just $\langle V \rangle$.

Induction hypothesis. For all (c_0, C)-colourful Rabin games using Steven strategy σ, in the restricted game $\mathcal{G}|_\sigma$, all fair paths satisfy the Rabin condition. Since we have fixed a strategy σ, to avoid cumersome notation, we henceforth refer to the game $\mathcal{G}|_\sigma$ using $\mathcal{G}$ itself, but require that all fair paths in $\mathcal{G}$ to satisfy the fair-Rabin condition. We assume that there is a (c_0, C)-colourful fair decomposition

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

where $c_0 \notin B_v$ for all $v \in V$, and for all $v \in V$, if $c_0 \in G_v$ then $v \in A$.

Induction step. Consider all vertices $B = \{v \mid c_0 \in G_v\}$, and let $A \supseteq B$ be the maximum set of vertices from which every fair path starting from A visits some vertex from B. It is routine to verify that there is such a unique maximum set.

Consider the subgame $\mathcal{G}_1$ induced by the set of vertices $V \setminus A$. This is a subgraph of $\mathcal{G}$ also satisfies the property that all vertices have an outgoing edge. More importantly, all fair paths in it satisfy the Rabin condition as these paths are also fair paths in the original graph. Furthermore, there are no vertices v such that $c_0 \in G_v$ or $c_0 \in B_v$ for $v \in W \setminus A$.

Consider an SCC decomposition of the graph induced by $V \setminus A$. Consider a bottom SCC (an SCC from which there is no path to other maximal SCCs) V_1 of the graph induced by $V \setminus A$. Consider a path π such that the set of all vertices visited by π infinitely often is exactly V_1. Clearly, there is no unfair path from V_1 to the subgame $\mathcal{G}_1 \setminus V_1$, since there is no path out of V_1. More specifically, the path that visits all the edges within V_1 infinitely often satisfies the fair Rabin condition. This implies that there is some colour c_1 such that $c_1 \notin B_v$ for all $v \in V_1$ and $c_1 \in G_v$ for some $v \in V_1$.

Therefore, by induction, there is a $(c_1, C \setminus \{c_1\})$-colourful fair decomposition of V_1, say $\mathcal{D}_1$. Let A_1 denote the maximum set of vertices in $V \setminus A$ from which all fair paths lead to a vertex in V_1.

Now consider the game $\mathcal{G}_2 = (V \setminus A) \setminus A_1$, which has fewer vertices and is a subgraph of $\mathcal{G}$ and where again there are no vertices v such that $c_0 \in G_v$ or $c_0 \in B_v$ for $v \in \mathcal{G}_2$, there must be a (c_0, C)-colourful fair decomposition.

Let this fair decomposition be:

$$\mathcal{D}' = \langle \varnothing, (c_2, V_2, \mathcal{D}_2, A_2), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

Observe that there are no vertices v where c_0 is a good colour or a bad colour for v. Therefore, the top set of vertices is $\varnothing$, due to our induction hypothesis.

We claim that the colourful fair decomposition

$$\mathcal{D} = \langle A, (c_1, V_1, \mathcal{D}_1, A_1), (c_2, V_2, \mathcal{D}_2, A_2), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \rangle$$

constructed from the sets defined above is a (c_0, C)-colourful decomposition.

It is routine to verify that the decomposition constructed satisfies all the properties of a colourful fair decomposition by construction. □

Lemma 10.2.3 ((2 $\implies$ 3) of Theorem 10.2.1)**.** *Given a (c_0, C)-colourful fair Rabin graph $\mathcal{G}$ with live edges L, and a (c_0, C)-colourful fair decomposition of it, there is an live-consistent λ from the vertices of such a game to a $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$.*

Proof. We follow suite of Lemma 9.2.3 in this proof, and combine it with a fair distance function. As in the previous proof, we first fix a positional strategy for Steven that ensures that he has a Steven decomposition $\mathcal{D}$ on the restricted game $\mathcal{G}|_\sigma$. Note that it is enough to treat the game as if all the vertices now belong to Audrey and then show that the mapping λ (that we shall define below) is a live-consistent mapping. Henceforth, we assume that $\mathcal{G}$ is the game obtained from fixing a Steven strategy.

To define such a mapping, we in turn use a *fair-distance function* for a fixed set of target vertices T, written D_T, and sometime just D when T is clear from context. Intuitively, such a function assigns to all vertices that can eventually reach T, a number natural number which indicates a modified distance in the game $\mathcal{G}$ from the target set. However, note that unlike a distance function defined for Lemma 9.2.3, this function does not decrease along all edges. In fact, it might increase along an edge of $\mathcal{G}$, if it has a live edge from the same vertex along which it

decreases. Let A be the set of vertices in V such that all fair paths in $\mathcal{G}$ eventually visits T. More formally, this function D_T maps the set of vertices A of the fair game $\mathcal{G}$, to a natural number from $\{0, 1, \ldots, |A| - 1\}$ such that

- if $v \in T$, then $D_T(v) = 0$;
- if there are no live edges from v, then $D_T(v) > D_T(w)$ for all $v \to w$;
- if there is a live edge from v, then $D_T(v) > D_T(w)$ for some live edge $v \to w$.

We prove the existence of such a function D_T in Proposition 10.2.4 after the proof of this lemma.

Suppose $C = \varnothing$. In this case, our $(c_0, \varnothing)$-colourful fair decomposition $\mathcal{D} = \langle V \rangle$. Let t denote the length of the longest path in $\mathcal{G}$ which does not visit a vertex for which c_0 is a good colour. Then we consider an $\mathbb{L}$-labelled (c_0, C)-colourful tree $\mathcal{L}$, which consists of t leaves, all denoted by $\{\langle \alpha_1 \bot \rangle, \ldots, \langle \alpha_t \bot \rangle\}$, where $\alpha_1 < \alpha_2 < \cdots < \alpha_t$, each α_i, an element of $\mathbb{N}$.

We construct the map λ that assigns all vertices $v \in V$ such that $c_0 \in G_v$ to the root of a labelled (c_0, C)-colourful tree denoted by $\langle\rangle$. For vertices v where $c_0 \notin G_v$, we define $D_T(v)$ as the fair-distance function to the set of vertices $T = \{v \in V \mid c_0 \in G_v\}$. We then define λ for each $v \in V \setminus T$ to be $\langle \alpha_i \bot \rangle$ where $D_T(v) = i$.

We verify that such a mapping satisfies the condition for it to be live-consistent. Since $B_v = \varnothing$ for all v, all the vertices satisfy B trivially. Observe that since u is a child of the root, all edges out of u are live-consistent. We only need to show that the vertex u itself is live-consistent. If there are no fair edges, then the arguments are similar to Lemma 9.2.3. If there is a live edge, showing vertex u is live consistent is routine and follows from the definition of fair distance and live consistent, and hence we omit it.

Suppose $C \neq \varnothing$. In this case, we have a (c_0, C)-colourful fair decomposition and

$$\mathcal{D} = \left\langle A, (c_1, V_1, \mathcal{D}_1, A_1), \ldots, (c_j, V_j, \mathcal{D}_j, A_j) \right\rangle.$$

Inductively, for each V_i, which has a $(c_i, C \setminus \{c_i\})$-colourful fair decomposition, we have a mapping λ_i to an $\mathbb{L}$-labelled $(c_i, C \setminus \{c_i\})$-colourful tree $\mathcal{L}_i$.

We give a Rabin measure λ into the colourful tree $\mathcal{T}$ defined as the set

$$\left\{ \left\langle \alpha_1^0 \bot \right\rangle, \ldots, \left\langle \alpha_t^0 \bot \right\rangle \right\} \bigcup_{i=1}^{j} \left\{ \alpha_0^i c_i \odot \mathcal{L}_i, \left\langle \alpha_1^i \bot \right\rangle, \ldots, \left\langle \alpha_t^i \bot \right\rangle \right\}$$

of nodes where α^i_ℓ are elements from $\mathbb{L}$ such that if $i_1 < i_2$, then $\alpha^{i_1}_\ell < \alpha^{i_2}_{\ell'}$ and if $\ell_1 \prec \ell_2$, then $\alpha^i_{\ell_1} \prec \alpha^i_{\ell_2}$. We define $\lambda(u)$ from a fair decomposition $\mathcal{D}$ above as follows. The tree is similar to the ones in Fig. 9.5.

- If $u \in A$ and $c_0 \in G_u$, then $\lambda(u) = \langle\rangle$.
- If $u \in A \setminus \{v \in A \mid c_0 \in G_v\}$, we define $\lambda(u) = \left\langle \alpha^0_\ell \bot \right\rangle$ where $D_B(u) = \ell$, with D_B defined as the fair-distance function to the target set $B = \{v \mid c_0 \in G_v\}$.
- For vertices $u \in V_i$, we define $\lambda(u) = \alpha^i_0 c_i \odot \lambda_i(u)$, where λ_i is obtained inductively.
- For vertices $u \in A_i \setminus V_i$, we define $\lambda(u) = \left\langle \alpha^i_\ell \bot \right\rangle$ where $D_{V_i}(u) = \ell$, where D_{V_i} is again defined as in Proposition 10.2.4 with target set V_i.

We show that the λ defined above satisfies the conditions required for it to be a live-consistent Rabin measure. To prove live consistency, we need to show that each vertex in the graph $\mathcal{G}$ is consistent or live consistent.

For the rest of the proof, we sometimes write A_0 to also refer to A and V_0 to refer to the set $\{v \in A \mid c_0 \in G_v\}$. Let $\mathcal{G}_i$ be defined similarly to the definition of a fair decomposition, where $\mathcal{G}_1 = V \setminus A$, and $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$. We moreover define $\mathcal{G}_0 = V$. The following observation about the map λ defined is useful to show that it is indeed a live-consistent map.

> ($) For $i \in \{0, 1, \ldots, j\}$, any vertex in $u \in V \setminus \mathcal{G}_i$ is such that $\lambda(u) \prec \lambda(v)$ for any $v \in \mathcal{G}_i$.

- If $u \in A$ and $c_0 \in G_u$, then $\lambda(u) = \langle\rangle$ and for all edges $u \to v$ satisfies $\mathsf{G}_\downarrow$. Since the root is coloured with c_0, such edges also satisfy B since $B_u = \varnothing$.
- If $u \in A$ and $c_0 \notin G_u$, then since $\lambda(u) = \left\langle \alpha^0_\ell \right\rangle$ where ℓ is the value given by the fair distance function (Proposition 10.2.4), either there are no live edges and instead all edges $u \to v$ satisfies $\mathsf{G}_\succ$ or there is at least one live edge and it satisfies $\mathsf{G}_\succ$ whilst all the other edges satisfy G_ℓ. Since the root is coloured with c_0, all edges from u also satisfies B since $B_u = \varnothing$.
- If $u \in A_i \setminus V_i$, for $i \in \{0, 1, \ldots, j\}$, and suppose $\lambda(u) = \left\langle \alpha^i_\ell \bot \right\rangle$ we show that u satisfies $\mathsf{G}_\succ$ or G_ℓ, and at least one live edge satisfies $\mathsf{G}_\succ$. Since u is mapped to a child of the root, all edges from it satisfy G_ℓ trivially. All we are left to show is that at least one edge satisfies $\mathsf{G}_\succ$whenever there is a fair edge from u. But this follows from the definition of λ, which was derived from the fair-distance

function, with only minor modifications needed from the similar case in the proof of Lemma 9.2.3.

That vertex u also satisfies B because the only ancestor of $\lambda(u)$ is $\langle\rangle$, and it is coloured with c_0, and $c_0 \notin B_v$ for any v, and therefore specifically $c_0 \notin B_u$.

- If $u \in V_i$ for $i \in \{1, \ldots, j\}$, for all edges $u \to v$, vertex v is either in V_i or in $V \setminus \mathcal{G}_i$. This is because, due to the definition of a fair decomposition, V_i is a set of vertices which has no fair path to $\mathcal{G}_i \setminus V_i$. If $v \in V \setminus \mathcal{G}_i$, we know that $\lambda(u) \succ \lambda(v)$ from (\$), and thus $\mathsf{G}_\succ$ is satisfied. On the other hand, if $v \in V_i$, then $\lambda_i(u)$ and $\lambda_i(v)$ are both defined. If edge $u \to v$ satisfies $\mathsf{G}_\downarrow$ in λ_i, then it continues to satisfy $\mathsf{G}_\downarrow$ in λ. In the case where edge $u \to v$ satisfies $\mathsf{G}_\succ$ in λ_i, that is, $\lambda_i(u) \succ \lambda_i(v)$, then it satisfies $\mathsf{G}_\succ$ in λ as well. Finally, again for G_ℓ, the same holds inductively. We end the proof by remarking that $\mathsf{ColourSet}(\lambda(u)) = \mathsf{ColourSet}(\lambda_i(u)) \cup \{c_i\}$. Thus, B is also satisfied by the edge $u \to v$. □

Proposition 10.2.4. *Consider a graph $\mathcal{G}$ with live edges L, and a target set T. Let A be a set of vertices such that all fair paths that start from A, stay in A and lead to a vertex in T, then there is a function D_T from all vertices of a A to $\{0, 1, \ldots, |A|-1\}$ such that $D_T(u)$*

- *if $v \in T$, $D_T(u) = 0$;*
- *if there are no live edges from v, then $D_T(u) > D_T(v)$ for all $u \to v$;*
- *if there is a live edge from u, $D_T(u) > D_T(v)$ for some live edge $u \to v$.*

Proof. Consider the set consisting of all functions D from A to $\mathbb{N} \cup \{\infty\}$ that satisfy

- if $v \in T$, $D(u) = 0$;
- if there are no live edges from v, then $D(u) > D(v)$ for all $u \to v$;
- if there is a live edge from u, $D(u) > D(v)$ for some live edge $u \to v$.

It can be verified that this set of functions is closed under point-wise minimum. Therefore, this set indeed has a smallest element. Consider the smallest such function D_T that satisfies the above condition. We claim indeed that the smallest function can be defined as

$$D_T(u) = \begin{cases} 0 & \text{if } v \in T \\ \max\{D_T(v) \mid u \to v \in E\} & \text{if there are no live edges from } v \\ \min\{D_T(v) \mid u \to v \in L\} & \text{if there is some live edge from } v. \end{cases}$$

Indeed, it is clear that any function that does not satisfy the above condition at some vertex can be modified locally to produce a smaller function that does satisfy this condition at a specific vertex. First we show that such a function D_T maps no vertex to ∞ if all fair paths lead to T.

Assume to the contrary that the set of all vertices labelled above with ∞, denoted by the set U is non-empty. For each vertex $u \in U$, if there are no outgoing live edges from u, then there is some edge $u \to v$ such that $v \in U$. If there are live edges, all outgoing live edges also lead to U, since $D_T(u) = \min\{D_T(v) \mid u \to v \in L\}$.

Hence any path in $U \subseteq A$ can be extended infinitely to remain in U in such a way that is fair, contradicting our assumption that all fair paths in A lead to T. The minimality of D_T also ensures that the range of D_T does not exceed $|A|-1$. □

Lemma 10.2.5 ((3 $\implies$ 1) of Theorem 10.2.1)**.** *Given a (c_0, C)-colourful Rabin graph $\mathcal{G}$ with a designated set of live edges L, if there is a live-consistent λ from vertices of $\mathcal{G}$ to a $\mathbb{L}$-labelled (c_0, C)-colourful tree then all infinite paths satisfy the fair Rabin condition.*

Proof. We need to show that all infinite path that are fair also satisfies the Rabin condition in this graph where each vertex is live-consistent or consistent. The rest of the proof is similar to the proof of Lemma 9.2.6, with modifications taken into account for fairness.

For an infinite fair path $\rho = v_0 \to v_1 \to v_2 \to \ldots v_i \to v_{i+1} \to \ldots$ consider $\lambda(\rho)$, the infinite sequence $t_0, t_1, \ldots$ where each $t_i = \lambda(v_i)$. We define $t_{\min}$ as the smallest element that occurs infinitely often among the sequence consisting of the elements t_i and $c = \mathsf{colour}(t_{\min})$. We show that this fair run satisfies the Rabin condition with the following three items.

(a) $c \neq \bot$;

(b) $c \in G_v$ for infinitely many v_is;

(c) $c \notin B_v$ for all but finitely many v_is.

We will prove the items in Proposition 9.2.5 for the path ρ, which will require slight changes in the argument due to the newly introduced G_ℓ condition. That is, we will prove $t_{\min}$ is

1. the largest common ancestor of t_i and t_{i+1} infinitely often

2. an ancestor of all but finitely many t_is.

Notice that assuming that the above two items hold for $t_{\min}$, Item a follows from $t_{\min}$ being an ancestor, and Item b and Item c follow directly from Item 1 and Item 2, respectively. To prove Item 1 and Item 2, we start by recalling that for any $v_i \to v_{i+1}$ one of the following conditions is true, since all vertices are consistent:

(i) $\lambda(v_i) \succ \lambda(v_{i+1})$ (the edge satisfies $\mathsf{G}_{\succ}$);

(ii) $\lambda(v_i) = \mathsf{GCA}(\lambda(v_i), \lambda(v_{i+1}))$ (the edge satisfies $\mathsf{G}_{\downarrow}$);

(iii) $\mathrm{parent}(\lambda(v_i)) = \mathsf{GCA}(\lambda(v_i), \lambda(v_{i+1}))$, but there is some a $v_i \to w \in L$ such that $\lambda(v_i) \succ \lambda(w)$ (this edge satisfies G_{ℓ} but some other live-edge out of it however satisfies $\mathsf{G}_{\succ}$);

Let p be the position after which all $k > p$, the node t_k are such that t_k occurs infinitely often in ρ. Let this position p be chosen moreover so that $t_p = t_{\min}$.

1. For any i with $\lambda(v_i) = t_{\min}$, observe that the edge (v_i, v_{i+1}) cannot satisfy condition (i), since $t_{\min}$ is the smallest element occurring infinitely often and therefore $\lambda(v_i) \not\succ \lambda(v_{i+1})$. This edge cannot satisfy condition (iii), because since the infinite path is fair, this would imply there exists some outgoing edge of v_i which satisfies $\mathsf{G}_{\succ}$which is taken infinitely often, contradicting the minimality of $t_{\min}$. Therefore, on path ρ, an edge (v_i, v_{i+1}) with $\lambda(v_i) = t_{\min}$ satisfying $\mathsf{G}_{\downarrow}$ should be taken infinitely often. Item 1 directly follows from this argument.

2. We argue that t_p is an ancestor of all t_j with $j \geqslant p$. Let the next occurrence of $t_{\min}$ be at t_q. More formally, we show $t_p = \mathsf{GCA}(t_p, t_p) = \mathsf{GCA}(t_p, t_{p+1}) = \cdots = \mathsf{GCA}(t_p, t_q)$. We proceed by induction. In the base case $\mathsf{GCA}(t_p, t_p)$ is trivially t_p. We assume that t_p is an ancestor of t_{p+i}, and show t_p is also an ancestor of t_{p+i+1}. For brevity, we use $j = p + i$. For any two consecutive t_j, t_{j+1}, either $t_j \succ t_{j+1}$ (by condition (i)), t_j is an ancestor of t_{j+1} (by condition ii) or $\mathrm{parent}(t_j)$ is an ancestor of t_{j+1} (by condition (iii)). In the first two cases, we get t_p is an ancestor of t_{j+1} following the proof of Proposition 9.2.5. In the last case, we observe that $t_j \neq t_p = t_{\min}$ from the minimality of $t_{\min}$ as shown in the proof of item 1. Hence t_p is an ancestor of the parent of t_j. Since the parent of t_j is an ancestor of t_{j+1} from condition (iii), any other ancestor of the parent of t_j is an ancestor of t_{j+1} as well. Thus we get $t_p = \mathsf{GCA}(t_p, t_{j+1})$. This proves our claim and item 2 follows directly. □

10.3 A lifting algorithm for fair Rabin games

We show that the lifting algorithm described in Chapter 9 can be modified to accommodate edges with strong fairness condition on Audrey's vertices. The modifications are minimal and require only one additional condition for a lift operator to perform.

Lift operator for Rabin games with liveness. We extend the definition of the lift operator in the setting of games with live edges. Consider a mapping λ from vertices of a game to $\mathcal{L}^\top$, an $\mathbb{L}$-labelled (c_0, C)-colourful tree rooted at colour c_0 with an additional $\top$ element, i.e. $\mathcal{L}^\top$.

For an edge $e = u \to v$ in the game $\mathcal{G}$, we define $\text{fairlift}_\lambda(v)$ to be the smallest element $t \in \mathcal{L} \cup \{\top\}$ at least as large as $\lambda(v)$ in the mapping $\lambda[v \coloneqq t]$ and such that v is consistent or live-consistent.

LiftFair_v to be a function from the set of all maps from V to $\mathcal{L} \cup \{\top\}$ to itself such that

$$\text{LiftFair}_v(\lambda)(u) = \begin{cases} \lambda(u) & \text{if } u \neq v \\ \text{fairlift}_\lambda(v) & u = v \end{cases}$$

Proposition 10.3.1. *The function* LiftFair_v *is inflationary and monotone.*

By definition this operation is inflationary, since LiftFair_v on a mapping λ is equal at all $u \neq v$ and at least as large as the $\lambda(v)$ at v.

To show that the above function is monotonic, we show that for two mappings that satisfy $\lambda_1 \sqsubseteq \lambda_2$, we also have $\text{LiftFair}_v(\lambda_1) \sqsubseteq \text{LiftFair}_v(\lambda_2)$. This follows closely from the definition of fairlift.

Since the operator LiftFair_v is both inflationary and monotonic, the simultaneous fixpoint of this operator exists, and moreover any maximal chain obtained by application of these operators on λ reaches the least simultaneous fixpoint larger than λ.

To conclude that a lifting algorithm would take time $\mathcal{O}(mk|\mathcal{L}| \lg n \lg k)$, we only need to show that $\text{fairlift}_\lambda(v)$ can be computed in time $\mathcal{O}(\deg(v)k \lg n \lg k)$, which we do so later in Lemma 10.3.2. But we state our main theorem, which we obtain from Theorem 10.2.1, Lemma 10.3.2 and Theorem 9.4.2.

Theorem H. *Finding the winner in a fair Rabin game can be determined in*

$\mathcal{O}(nk \log n \log k)$ *space and time*

$$\widetilde{\mathcal{O}}\left(nm \cdot k! \left(\min\left\{n2^k, \binom{\ell+k}{k-1}\right\}\right)\right).$$

We finally show the building block of the fair lifting operator can be implemented in time $(\deg(v)k\ell \lg k)$ below.

Lemma 10.3.2. *Given a mapping λ, a (c_0, C)-colourful Rabin game $\mathcal{G}$ with n vertices, along with a vertex v, computing* $\mathrm{fairlift}_\lambda(v)$ *takes time $\mathcal{O}(\deg(v)k\ell \lg k)$ with k colours and $\ell = \lceil \lg n \rceil$.*

Proof. Suppose there are no live edges from a vertex v, observe that $\mathrm{LiftFair}_v = \mathrm{Lift}_v$, defined in Section 9.3. Otherwise if v is an Audrey vertex that has live outgoing edges, we want to show how to calculate $\mathrm{fairlift}_\lambda(v)$, the minimum value larger than $\lambda(v)$ that makes v consistent, or live-consistent with respect to λ. To do this, we will get the minimum of two values, the one obtained just from lift and the other from fairlift. Since computing $\max_{v \to u}\{\mathrm{lift}_\lambda(v, u)\}$ already takes the claimed running time from Lemma 9.4.9, setting v to it in λ would make it consistent and we dedicate to the rest of the proof to finding the time taken to compute the other where setting v to it makes the map live-consistent.

Suppose $u_1, u_2, \ldots, u_d$ are the out-neighbours of the vertex v, such that $\lambda(u_1) \prec \lambda(u_2) \prec \cdots \prec \lambda(u_d)$. We assume strict inequality, because it is enough to consider only one among two edges with the same value. We assume that v is not already live-consistent with respect to λ.

Recall that our objective is to find a node larger than $\lambda(v)$, such that mapping v to this node would make v live-consistent. Moreover, at least one edge needs to satisfy $\mathsf{G}_\succ$ for v to be live-consistent.

Let i be the smallest index such that $v \to u_i$ is live.

1. If $\lambda(v) \preceq \lambda(u_i)$, then there are no outgoing edges from v that satisfy $\mathsf{G}_\succ$. Therefore to make v live-consistent, it must take a value at least as large as the next element of $\lambda(u_i)$.

2. If $\lambda(v) \succ \lambda(u_i)$, the live-edge $v \to u_i$ already satisfies $\mathsf{G}_\succ$ with respect to λ and any increase in the value of v would satisfy $\mathsf{G}_\succ$ too. In this case, we find the largest j such that $\lambda(v) \succeq \lambda(u_j)$. Since we assumed v is not consistent, $j < d$.

For case 1 above, we set m to be i and for case 2, we let m to be j. Since v must be set to a value larger than $\lambda(u_i)$ for item 1 and $\lambda(v)$ for item 2, we declare

$t = \lambda(u_m)$ in the former and $t = \lambda(v)$ in the latter case. For both cases, we have $\lambda(u_1) \prec \cdots \prec \lambda(u_m) \preceq t \prec \lambda(u_{m+1}) \prec \lambda(u_d)$.

Let $t_1, \ldots, t_{d-m}$ correspond to the sequence $\lambda(u_{m+1}), \ldots, \lambda(u_d)$. The problem can now be solved in $\mathcal{O}(\deg(v)k\ell \lg k)$, thanks to Proposition 10.3.3. We remark that the reason for i in the third item being strictly smaller than p is that if indeed $t^* \succcurlyeq t_p$, then all edges must satisfy $\mathsf{G}_\succ$ and this case is similar to computing $\mathrm{lift}_\lambda(v)$. Proving the proposition in the following, we conclude that $\mathrm{fairlift}_\lambda(v)$ can be computed in $\mathcal{O}(\deg(v)k\ell \lg k)$. □

Proposition 10.3.3. *In the n-universal (c_0, C)-colourful tree $\mathcal{L}_C^\ell$ constructed (preceding Lemma 9.4.8) and given subset of colours B_v and a sequence $t \prec t_1 \prec \cdots \prec t_p$, finding the smallest such node t^*, if one exists, that satisfies:*

- $t^* \succ t$
- *coloured with $\perp$;*
- $\mathsf{ColourSet}(t^*) \cap B_v = \varnothing$;
- *assuming $t_0 = t$, there is some $0 \leqslant i < p$ such that $t^* \succ t_0, \ldots, t_i$ and the parent of t^* is an ancestor of each of $t_{i+1}, \ldots, t_p$;*

takes time proportional to $\mathcal{O}(pk\ell \lg k)$.

Proof. We use the following property of the constructed universal tree $\mathcal{L}_C^\ell$. For any node in the tree, if its colour is not $\perp$, then it has a child coloured with $\perp$. Moreover, any element $(\omega_1 c_{i_1}, \ldots, \omega_m c_{i_j})$ coloured with $\perp$, i.e., $c_{i_j} = \perp$ is larger than the above node but $c_{i_j} \neq \perp$.

Let t' be the largest common ancestor of vertices $t_1, \ldots, t_p$. The following are the exhaustive list of cases to consider.

- If $t \succ t'$, we get can deduce from Proposition 9.2.4 that t_1 is a strict descendant of t which in turn is a strict is a descendent of t'. Here, if $\mathsf{ColourSet}(t') \cap B_v \neq \varnothing$, then no such t^* exists. If on the other hand $\mathsf{ColourSet}(t') \cap B_v = \varnothing$, then t^* is the smallest child of t' that is larger than t coloured with $\perp$. We remark that such a t^* exists, because we can show by Proposition 9.2.4 that t is a strict ancestor of t_1.
- If $t \preceq t'$ and let $\hat{t}$ be a common ancestor of t and t'. We list all the descendants of $\hat{t}$ that are also ancestors of the largest of the input vertices t_p. Let these be in order : $\hat{t} = t^0 \prec t^1 \prec \cdots \prec t^q = t_p$.

Since we want $\mathsf{colour}(t^*) \cap B_v = \varnothing$, we want $\mathsf{colour}(\text{parent}(t^*)) \cap B_v = \varnothing$. Notice that the parent of t^* is among the sequence $t^0 \prec t^1 \prec \ldots \cdots \prec t^q$. For each of these nodes t^j among $t^0, t^1 \ldots t^{q-1}$, in this order, we perform the following checks:

- Is $\mathsf{colour}(t^j) \cap B_v = \varnothing$? If not, we declare that no such t^* exists with any other $t^{\geqslant j}$ as a parent.
- Find smallest child of t^j coloured with $\perp$. Except in the case of t^0, where we find smallest child of t^0 coloured with $\perp$ and is larger than t, if one exists. We declare it say t^j_*.
- We return the smallest such t^i_* found.

This computation as shown above has three steps. Firstly, finding the least common ancestor of p nodes. Secondly, we find next child of certain nodes coloured with $\perp$. Finally we take the minimum of at most k nodes. The first of these computations is the costliest and takes time $\mathcal{O}(pk\ell \log k)$. The next node that is coloured with $\perp$ is just a simple version of next^ℓ_C defined for the tree, where $C = \varnothing$ and therefore can be computed in time $\mathcal{O}(k\ell \log k)$. We compare and take minimum of nodes, which does not take more than time $\mathcal{O}(pk\ell \log k)$ if we compute the minimum on-the-fly. Moreover, since the vertices for which we are computing next is an increasing chain of tuples, we can implement the above in time proportional to $\mathcal{O}(pk\ell \log k)$. We remark however that this runtime assumes that we compute the common ancestors and required nodes once and store it for easy access to avoid an extra factor of p to our computation costs. □

10.4 Almost-sure winning stochastic Rabin games

Stochastic Rabin games are two player games with some added probabilistic states introduced in the game. The game arena is still a (c_0, C)-colourful Rabin graph. However, the vertices are partitioned into three sets V_A, V_S and V_R. The partition V_R intuitively corresponds to stochastic choices, where each edge is chosen uniformly at random.

We say that a positional strategy σ is almost-surely winning if the measure of all the paths in the Markov decision process $\mathcal{G}|_\sigma$ obtained satisfying the Rabin condition is 1. Although maximising the exact probability of winning in a stochastic Rabin game (maximising the measure of all paths obeying a strategy and satisfying the Rabin condition) might require non-positional strategies, it was shown in the work of Chatterjee, Henzinger and Jurdziński [CHJ05] for stochastic parity games,

and later generalised in the work of Chatterjee, de Alfaro and Henzinger [CdAH05] to stochastic Rabin games that positional strategies are enough for Steven to ensure an *almost-sure* winning play (winning with probability 1). In their work, Chatterjee, de Alfaro and Henzinger [CdAH05] gave constructions that modify any stochastic Rabin game with at most n vertices into a Rabin game with $\mathcal{O}(n+ks)$ many vertices where there are no vertices belonging to the random player, and the number of colours remains the same.

Corollary 10.4.0.1. *Deciding if Steven almost-surely wins a stochastic Rabin game takes time*

$$\widetilde{\mathcal{O}}\left(mnk^2k!\min\left\{n2^k,\binom{\lceil\lg n\rceil+k}{k-1}\right\}\right)$$

and $\mathcal{O}(nk\log n\log k)$ *space.*

The proof of the above corollary follows from the fact that a stochastic Rabin game can be turned into a Rabin game under strong transition fairness by declaring the random player vertices in the first one as Audrey vertices in the second, and turning all outgoing edges of random player vertices to live edges. The almost-sure winning region of Steven in the first game is equivalent to the winning region of Steven in the second.

We remark that a similar approach also gives a progress measure based algorithm parameterised by trees also for fair parity games, defined analogously to fair Rabin games. This was a model considered in the recent work of Sağlam and Schmuck [SS23], where they modify McNaughton- Zielonka's algorithm to solve such games. Our result for fair Rabin games can also be adapted to solve fair parity games using a progress measure algorithm by substituting our colourful universal trees instead with Jurdziński-Lazić universal trees.

Chapter 11

Lower bounds for solving Rabin games

Using our combinatorial construction of colourful universal trees, which accommodates any colourful decompositions, we significantly reduced our running time as well as state space complexity of algorithms to solve Rabin games in Chapter 9. Our algorithm took time linear in $k!^{o(1)}$ and a polynomial in n whilst requiring only $\widetilde{O}(nk)$ space. But one might wonder if this algorithm could be further improved, with this dependence on $k!$ reduced to at least a 2^k. However, the work of Calude, Jain, Khoussainov, Li, and Stephan [CJK+22] swiftly put an end to such hopes due to their complexity lower bounds subject to Exponential Time Hypothesis (ETH, the assumption that there exists $\delta > 0$ such that 3SAT problem cannot be solved in time $\mathcal{O}(2^{\delta n})$) for Rabin games. They proved that assuming ETH, there are no algorithms with running time $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$ for Rabin games, essentially giving a negative answer to our question. This reduction provided by Calude et al. starts with the Dominating Set problem and is rather involved.

We provide a simple reduction that reproves the tight complexity lower bound for solving Rabin games that follows from the work of Calude et al. More precisely, we prove that assuming ETH, there is no algorithm for this problem with running time $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$. The same lower bound for (the more general) Muller games follows as a direct corollary. By a minor twist of our construction, we can also reprove the lower bound for d-dimensional parity games reported by Calude et al. These games were studied by Chatterjee, Henzinger and Piterman [CHP07], and are a generalisation of parity games to different dimensions, therefore also known as generalised parity games.

We believe that our reduction for Rabin games is significantly simpler and

more transparent than that of Calude et al. but more importantly, it gives a better insight into the origin of the $2^{o(k \log k)}$ factor in the complexity of the problem. Analysing our algorithms from Chapter 9, this factor stems from considering all possible permutations of the k colours involved in the winning condition. In our reduction, those permutations form the space of potential solutions of a carefully chosen pivot problem —Permutation SAT, a special case of a temporal constraint satisfaction problem— which we discuss below.

Temporal CSPs and Permutation SAT. A constraint satisfaction problem (CSP) is the problem of deciding if there exists a variable assignment that satisfies a given set of constraints. *Temporal problems* is a rich family of CSPs that model planning various events on a timeline. In a basic form, every variable corresponds to an event that needs to be scheduled at some point of time and constraints speak about some events being in specific order (e.g., one preceding another), at the same time, or at different times. This is usually modelled with $\mathbb{Q}$ as the domain and constraints having access to predicates $<$, $\leqslant$, $=$, and $\neq$. A P vs NP dichotomy for finite languages within this formalism has been provided by Bodirsky and Kára [BK10].

Exponential Time Hypothesis. The Exponential Time Hypothesis is a complexity assumption introduced by Impagliazzo, Paturi and Zane [IPZ01] that postulates the following: there exists $\delta > 0$ such that the 3-SAT problem cannot be solved in time $\mathcal{O}(2^{\delta n})$, where n is the number of variables of the input formula. We refer the reader to the book by Cygan et al. [CFK$^+$15, Chapter 14] for an introduction to the applications of ETH for lower bounds within parameterized complexity.

Generalised parity objective. Generalised parity games were first considered in the work of Chatterjee, Henzinger, and Piterman [CHP07]. In a d-dimensional k-parity condition, each vertex is labelled with a d-dimensional vector of integers from $\{1, \ldots, k\}$. An infinite play satisfies this objective for Steven if and only if there is some coordinate such that the highest number that occurs infinitely often at this coordinate is even. Audrey wins otherwise.

11.1 Permutation SAT

Fix integers $\alpha \geqslant 2$ and $\beta \geqslant 1$ and let X be a finite set of real-valued variables. An *α-literal* is a predicate of the form $x_1 < x_2 < \ldots < x_{\alpha'}$ (being a shorthand for $(x_1 < x_2) \wedge (x_2 < x_3) \wedge \ldots \wedge (x_{\alpha'-1} < x_{\alpha'})$) for some $2 \leqslant \alpha' \leqslant \alpha$ and variables

$x_1, x_2, \ldots, x_{\alpha'}$ belonging to X; a *literal* is a 2-literal (i.e., a predicate of the form $x_1 < x_2$). An (α, β)-*clause* is a disjunction of at most β α-literals, and an (α, β)-*formula* is a conjunction of (α, β)-literals. By β-clauses and β-formulas we mean $(2, \beta)$-clauses and $(2, \beta)$-formulas, respectively.

If ϕ is a formula with variable set X, then for a permutation π of X we define the satisfaction of (literals and clauses of) ϕ by π in the obvious manner. In the (α, β)-Permutation SAT problem we are given an (α, β)-formula ϕ and the task is to decide whether there exists a permutation of the variables of ϕ that satisfies ϕ. β-Permutation SAT is a shorthand for $(2, \beta)$-Permutation SAT.

In this section we prove the following hardness result.

Theorem 11.1.1. *Assuming ETH, there is no algorithm for* 4-Permutation SAT *that would work in time* $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$*, where* k *is the number of variables and* n *is the number of clauses.*

To prove Theorem 11.1.1 we use the problem $k \times k$-Clique considered by Lokshtanov, Marx, and Saurabh [LMS18]. They showed that, unless ETH fails, this problem cannot be solved in $2^{o(k \log k)}$-time. We first define $k \times k$-Clique below, and then reduce $k \times k$-Clique to 4-Permutation SAT.

An instance of the $k \times k$-Clique problem is an undirected graph $\mathcal{G}$ with the vertex set $\{1, \ldots, k\} \times \{1, \ldots, k\}$ (that we can represent as a grid). This graph $\mathcal{G}$ is a positive instance of $k \times k$-Clique if there are k vertices chosen such that there is exactly one vertex from each *row* of the grid that forms a k-clique, that is, a k-clique in which no two vertices share the same first component.

Theorem 11.1.2 ([LMS18, Theorem 2.4])**.** *Assuming ETH, there is no* $2^{o(k \log k)}$*-time algorithm for* $k \times k$-Clique*.*

The reduction. We now reduce $k \times k$-Clique to 4-Permutation SAT. Let $\mathcal{G}$ be an instance of $k \times k$-Clique. Given $\mathcal{G}$, we construct a 4-formula $\phi_{\mathcal{G}}$ over variable set $X \coloneqq \{x_1, \ldots, x_k, x_{k+1}, y_1, \ldots, y_k\}$ as follows

Recall that the vertices of the graph $\mathcal{G}$ are of the form (i, j) for $i, j \in \{1, \ldots, k\}$. We say that vertex (i, j) is in the i^{th} row and j^{th} column. To construct $\phi_{\mathcal{G}}$, we first write the following $3k$ many 1-clauses:

$$x_1 < x_2, \quad x_2 < x_3, \quad \ldots, \quad x_k < x_{k+1},$$
$$x_1 < y_1, \quad x_1 < y_2, \quad \ldots, \quad x_1 < y_k$$
$$y_1 < x_{k+1}, \quad y_2 < x_{k+1}, \quad \ldots, \quad y_k < x_{k+1}$$

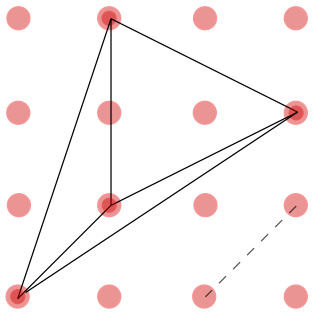


Figure 11.1: The construction in Section 11.1.

The disjunction of these clauses ensures that in any permutation satisfying $\phi_{\mathcal{G}}$, the variables $x_1, \ldots, x_{k+1}$ are ordered exactly in this way, while variables $y_1, \ldots, y_k$ are sandwiched between x_1 and x_{k+1}.

Next, we introduce clauses that restrict the placement of variables $y_1, \ldots, y_k$ within the chain $x_1 < x_2 < \ldots < x_{k+1}$. The intention is the following: placing y_i between x_j and x_{j+1} corresponds to choosing the vertex (i, j) to the clique. Hence, it remains to introduce clauses ensuring that vertices chosen in this way in consecutive rows are pairwise adjacent. To this end, for every pair $(a, b), (c, d)$ of vertices non-adjacent in $\mathcal{G}$, we construct the 4-clause

$$(y_b < x_a) \vee (x_{a+1} < y_b) \vee (y_d < x_c) \vee (x_{c+1} < y_d).$$

Note that logically, this 4-clause is equivalent to

$$\neg\left((x_a < y_b < x_{a+1}) \wedge (x_c < y_d < x_{c+1})\right).$$

Thus, intuitively speaking, the 4-clause forbids simultaneously choosing (a, b) as well as (c, d) to form a clique.

Consider the clique in Fig. 11.1. It corresponds to the permutation $x_1 < y_4 < x_2 < y_1 < y_3 < x_3 < x_4 < y_2 < x_5$ (with y_1 and y_3 possibly swapped). Note that the vertex $(2, 2)$ is *not* chosen in the clique. The dashed non-edge $((4, 3), (3, 4))$ is disallowed by the clause $\neg\left((x_4 < y_3 < x_5) \wedge (x_3 < y_4 < x_4)\right)$ which ensures if y_4 appears between x_3 and x_4, then y_3 does not appear between x_4 and x_5. This concludes the construction of the formula $\phi_{\mathcal{G}}$. It remains to verify the correctness of the reduction.

Lemma 11.1.3. *The graph $\mathcal{G}$ admits a k-clique with one vertex from each row if and only if $\phi_{\mathcal{G}}$ is satisfiable.*

Proof. First suppose $\mathcal{G}$ contains a k-clique $K = \{(1, b_1), \ldots, (k, b_k)\}$. Consider any

permutation π of X such that

- $x_1 < x_2 < \cdots < x_k < x_{k+1}$, and
- $x_{b_i} < y_i < x_{b_i+1}$, for all $j \in \{1, \ldots, k\}$.

(Note that π is not defined uniquely, the relative placement of y_i and $y_{i'}$ can be arbitrary whenever $b_i = b_{i'}$.) It can be easily seen that K being a clique, implies that all clauses in $\phi_{\mathcal{G}}$ are satisfied. The 1-clauses are satisfied trivially, while every 4-clause constructed for a non-adjacent $(b, a), (d, c)$ is satisfied because (b, a) and (d, c) cannot simultaneously belong to K.

Suppose now that there is an ordering of X that satisfies $\phi_{\mathcal{G}}$. Clearly, it must be the case that $x_1 < x_2 < \cdots < x_k < x_{k+1}$. Further, for every $i \in \{1, \ldots, k\}$ we have $x_1 < y_i < x_{k+1}$, hence there exists j_i such that $x_{j_i} < y_i < x_{j_i+1}$. We let $K := \{(i, j_i) : i \in \{1, \ldots, k\}\}$; note that K contains one vertex from each row. We claim that K is a clique in $\mathcal{G}$. Indeed, since in $\phi_{\mathcal{G}}$ there is a clause disallowing that $((x_a < y_b < x_{a+1}) \wedge (x_c < y_d < x_{c+1}))$ whenever there is no edge between (a, b) and (c, d), all vertices of K must be pairwise adjacent. □

This concludes the proof of Theorem 11.1.1. We remark that establishing the complexity of 2- and 3-Permutation SAT remains an interesting and challenging open problem. Eriksson, in his MSc thesis [Eri19], shows that 2-Permutation SAT can be solved in time $((k/2)!)^2 \cdot (k+n)^{\mathcal{O}(1)}$, which gives roughly a $2^{k/2}$ multiplicative improvement over the naive algorithm.

11.2 Lower bound for Rabin games

Finally, in this section, we prove the main result of this chapter, stated as Theorem I below.

Theorem I ([CJK+22]). *Assuming the Exponential Time Hypothesis, there is no algorithm that solves Rabin games with n vertices and with k colours in time $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$.*

As mentioned before, we reduce from 4-Permutation SAT.

The reduction. Let $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be an instance of 4-Permutation SAT over k variables $\{y_1, \ldots, y_k\}$, where $C_1, \ldots, C_m$ are 4-clauses. We construct a (c_0, C)-colourful game $\mathcal{G}$ for $C = \{c_1, \ldots, c_k\}$ such that there is a strategy for Steven iff ϕ is satisfiable.

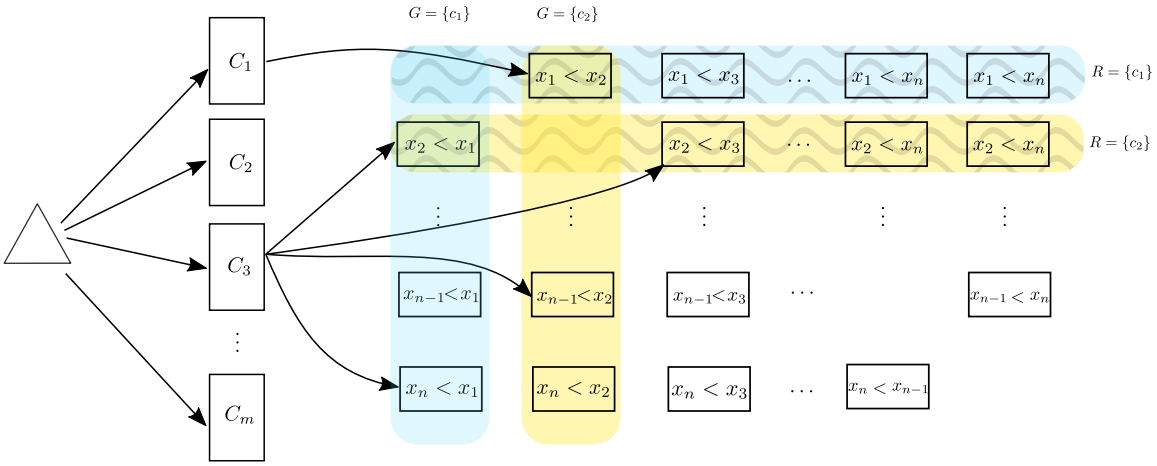


Figure 11.2: The construction in Section 11.2.

We first define the game graph $\mathcal{G}$; see Figure 11.2. There is an initial vertex Δ, as well as vertices $[C_1], \ldots, [C_m]$, one for each of the m 4-clauses in ϕ. Further, for each possible literal $x_i < x_j$, where $i, j \in \{1, \ldots, k\}$ and $i \neq j$, there is a vertex $[x_i < x_j]$. Vertex Δ belongs to Audrey, while all other vertices belong to Steven.

The intention is that whenever Audrey moves the token currently placed at Δ, she chooses a clause that she wishes to see satisfied. To facilitate this, we add edges $\Delta \to [C_\ell]$ for all $\ell \in \{1, \ldots, m\}$. Once the token is at a vertex $[C_\ell]$, Steven needs to respond with a literal present in C_i; the intention is that should be a literal true in C_i. Therefore, for every clause C_ℓ and literal $x_i < x_j$ present in C_ℓ, we add the edge $[C_\ell] \to [x_i < x_j]$. Finally, to allow Audrey checking further clauses, we add edges back to Δ: for every literal $x_i < x_j$, there is an edge $[x_i < x_j] \to \Delta$.

Next, we define the good and bad colours constituting the winning condition. For each vertex v *not* of the form $[x_i < x_j]$, we declare G_v as well as B_v to be empty, but for vertices $[x_i < x_j]$, we set

$$G_{[x_i<x_j]} = \{c_j\} \qquad \text{and} \qquad B_{[x_i<x_j]} = \{c_i\}.$$

Before we proceed to the formal verification of the correctness of the reduction, let us give some intuition. It is easy to see that, on the third turn, the token is placed at the vertex Δ. At each such moment, Audrey, in turn, chooses to move the token to any vertex corresponding to a clause C_ℓ, with the intention of challenging Steven about the satisfaction of C_ℓ. Then Steven has to declare the literal that satisfies C_ℓ. If Steven tries to "cheat" by picking literals that cannot be extended to a full ordering of the variables, then the winning condition is designed so that the

play will be losing for him. Consider the illustration in Fig. 11.2, where for example, ϕ of 4-PERMUTATION SAT which consists of m clauses such that the clause C_3 is $(x_1 < x_4) \vee (x_2 < x_1) \vee (x_2 < x_3) \vee (x_n < x_2)$ and C_1 has clause $(x_1 < x_2)$ only. Vertices that have c_1 or c_2 as good colours are highlighted using blue and yellow, respectively, and vertices with c_1 or c_2 colours as bad colours are again highlighted using blue and yellow, but with wavy lines added. Suppose Steven picks the vertices corresponding to the clause $(x_2 < x_1)$ from C_3 and $(x_1 < x_2)$ from C_1, then he loses the game, since Audrey would ensure both vertices $[x_2 < x_1]$ and $[x_1 < x_2]$ are seen infiniteley often. But observe that for both blue and red (c_1 and c_2), these vertices have both the colours as bad colours.

Lemma 11.2.1. *The instance ϕ of* 4-PERMUTATION SAT *is satisfiable if and only if Steven has a winning strategy in the constructed Rabin game.*

Proof. First suppose ϕ is satisfiable, therefore consider a satisfying permutation π. This gives rise to a (positional) winning strategy for Steven [EJ88, EJ99] such that in this Steven strategy, for each vertex $[C_\ell]$, there is an edge to the vertex $[x_i < x_j]$ that corresponds to a literal of C_ℓ that is satisfied in permutation π. Consider now any infinite play ρ for the game restricted to this strategy. Let L be the set of literals visited infinitely often by ρ, and let $i_{\max}$ be such that the variable $x_{i_{\max}}$ has the largest value in the permutation π among variables appearing in the literals of L. We argue that ρ satisfies constructed the Rabin condition with the colour $c_{i_{\max}}$ as a witness. This is because ρ visits $[x_i < x_{i_{\max}}]$ infinitely often (and therefore in L) for some i, and the colour $c_{i_{\max}} \in G_{[x_i < x_{i_{\max}}]}$, but for any vertex in $v \in L$, the colour $c_{i_{\max}} \notin B_v$ as ρ never visits any vertex $[x_{i_{\max}} < x_i]$ for any i ($x_{i_{\max}}$ has the largest value in the permutation).

Suppose now ϕ is not satisfiable. Then we need to show that for any positional strategy of Steven, Audrey can win with this strategy [EJ88, EJ99]. Indeed, consider a fixed positional strategy of Steven: for each Steven vertex $[C_\ell]$ the strategy contains an edge $[C_\ell] \to [x_{a_\ell} < x_{b_\ell}]$ for some literal $x_{a_\ell} < x_{b_\ell}$ appearing in C_ℓ. Since ϕ is not satisfiable, the set $\{x_{a_\ell} < x_{b_\ell} : \ell \in [m]\}$ of all selected literals has a cycle. That is, there are variables $x_{B_1}, \ldots, x_{B_p}$ such that literals $x_{B_1} < x_{B_2}, x_{B_2} < x_{B_3}, \ldots, x_{B_{p-1}} < x_{B_p}, x_{B_p} < x_{B_1}$ are among those selected by Steven's strategy. Observe now that for the fixed Steven's positional strategy, Audrey may play a strategy that repeatedly visits each of the vertices $[x_{B_1} < x_{B_2}], [x_{B_2} < x_{B_3}], \ldots, [x_{B_{p-1}} < x_{B_p}], [x_{B_p} < x_{B_1}]$ in a cycle, so that these are exactly the literal vertices visited infinitely often in the play. Then this play does not satisfy the constructed Rabin condition, since for each colour $c_i \in \{c_1, \ldots, c_k\}$, one of the following happens. For $i \in \{B_1, \ldots, B_p\}$, the colour c_i is a bad colour for

$[x_{B_p} < x_{B_i}]$ for $p = i - 1 \mod k + 1$. If, on the other hand, $i \notin \{B_1, \ldots, B_p\}$, then colour $c_i \notin G_v$ (or B_v) for any vertex in the above set of vertices is visited infinitely often. □

We end with the following corollary that claims that these lower-bound results also hold for Muller and generalised parity games.

Corollary 11.2.1.1. *Assuming the Exponential Time Hypothesis, there is no algorithm that solves*

- *Muller games k colours and n vertices,*
- *k-dimensional 3-parity games with n vertices, or*
- *2-dimensional k-parity games*

in time $2^{o(k \log k)} \cdot n^{\mathcal{O}(1)}$.

Using known reductions from Rabin games to Muller and k-dimensional 3-parity games, we obtain similar lower bounds as a corollary. We finally conclude by remarking that we can also extend our result to 2-dimensional k-parity games too. Indeed, consider the following assignment of colours to the same game graph $\mathcal{G}$: for each vertex of the form $[x_j < x_i]$, we assign the two-dimensional priority $(2j + 1, 2i)$ if $i < j$ and the priorities $(2j, 2i + 1)$ otherwise. The correctness of this reduction is similar to that for Rabin games presented above, and hence we leave the verification to the reader.

Chapter 12

Discussions and directions

This thesis has explored the landscape of solving complex two-player games on graphs, particularly those involving parity and Rabin objectives. Through a thorough examination of the structural properties and the development of innovative algorithms, we have made theoretical strides in solving such games.

The concept of attractor decomposition and the introduction of the Strahler number have proven to be pivotal in this thesis for solving parity games. This insight has laid the foundation for efficient algorithms for a range of parameters, made possible by our construction of succinct Strahler universal trees.

Our contributions were extended further through the formulation of new algorithms for solving parity games, which used a relaxation of attractor decompositions. Later, we solved Rabin games by extending the concept of attractor decompositions to include colourful decompositions.

As we conclude, we are not only enhancing our understanding of parity and Rabin games, but also offering valuable tools for addressing intricate challenges to solve these games. To further propel this inquiry, we leave the reader with some open ended discussions and questions to contemplate.

While we focus on the theoretical advancements throughout the thesis, an obvious future direction is to implement these algorithms. Lehtinen and Boker [LB20] had conjectured that parity games that arise in practice have small Strahler numbers. A natural question would be to verify their claim for parity games that arise from real-life scenarios. As a first step, we remark that it can be deduced from the work of Combes and Touati [CT20] that a randomly generated parity game (see their paper for an exact definition of what randomly generated means) has Strahler number 1.

Question I. *Do parity games that arise from practical applications or those avail-*

able in standard benchmarks have a low Strahler number?

We have constructed several algorithms that exploit the structure of attractor decompositions. Consequently, we believe that we should investigate the structure of ordered trees that arise from the attractor decompositions of hard examples and how they impact the intricate behaviour of our algorithms. With deeper insights, we foresee the potential to fine-tune these algorithms to create exceptionally efficient parity game solvers.

Question II. *What do the trees of attractor decompositions of these games look like? Are trees of attractor decompositions significantly smaller than the trees of progress measure algorithms for games that arise in practice?*

We also acknowledge that our symmetric algorithm can be less intuitive. When the input trees are smaller than the trees of attractor decomposition in the games, the sets returned may not provide complete information about the winning players within those sets. This limitation extends to other symmetric quasi-polynomial algorithms as well. Furthermore, in the context of synthesis, we ideally would like strategies to be produced alongside a solution. Understanding the sets returned by our algorithms better might lead to insights into how this algorithm can be simplified or made efficient.

Question III. *Can symmetric attractor-based algorithms, especially the quasi-polynomial ones (including ours) be modified to produce strategies for one or both players?*

Our algorithms for parity games are all parameterized by trees. It would be productive to understand the performance of these algorithms better on different families of parity games and for different universal trees. We do not have examples of families of games on which our symmetric algorithm takes more than polynomial time, even for our exponential version, which uses complete n-ary trees. A possible direction of research would be to show that our algorithm on complete trees takes quasi-polynomial, or some sub-exponential time. Conversely, if we can instead construct families of games for which this algorithm takes exponential time, we could shed new light on the central question of overcoming the quasi-polynomial barrier [CDF$^+$19].

Question IV. *Are there families of games for which our symmetric attractor-based algorithm (Algorithm 8 or Algorithm 9) requires more than polynomial amount of time?*

We developed the first polynomial space algorithm to solve Rabin games that is also fixed-parameter tractable with respect to the number of colours. Moreover, for LTL synthesis, existing tools, such as Rabinizer 4 [KMSZ18], convert LTL specifications to Rabin automata or parity automata. Analysing whether our algorithms for solving the obtained Rabin games outperform the approach of converting them to parity games and utilising our parity game algorithms or other parity game algorithms would be a fruitful pursuit.

Question V. *In practice, does our algorithm to solve Rabin games outperform the approach of converting them to parity games and using our algorithms to solve parity games?*

Our algorithm for both Rabin and fair Rabin games, like other progress measure algorithms, can display worst-case behaviour in certain asymmetric examples. To show that a vertex is losing for Steven, the measure needs to increase until it reaches $\top$. This lack of symmetry might lead to worst-case behaviour. But circumventing this problem by constructing similar measures for Audrey in the hopes of finding a symmetric algorithm is not as straightforward, as Audrey does not have a positional strategy in this game.

Question VI. *Are there algorithms that can solve Streett games and produce a strategy for Steven in time and space requirements similar to our algorithm based on colourful universal trees?*

On another tangent, symbolic algorithms for parity games are implicitly or explicitly guided by universal trees [Zie98, CDHS18, JMT22] constructed for both players. We believe that with some effort, our small colourful universal trees can be exploited to build symbolic algorithms that solve Rabin games or fair-Rabin games. For instance, one could draw inspiration from the Jurdziński-Morvan algorithm for parity games, combined with our construction of colourful universal trees. Indeed, we already have a definition of colourful decompositions, which one might hope to obtain as an end-result of such a recursive symbolic algorithm.

Question VII. *Can we construct a symbolic algorithm that solves Rabin games proportional to the size of our colourful universal trees?*

One can also analyse parameters with respect to which Rabin games are FPT. However, certain parameters have already been excluded—subject to conditions like $\mathsf{P} \neq \mathsf{NP}$ or ETH—due to established lower bound results [EJ99, GLM$^+$15], including our own result in Chapter 11. These include tree-width (of the underlying undirected graph), entanglement, DAG-width and any other directed width

measure, as the graph we produced has a constant value of all the above measures. Some work has already been done towards this direction for parity games [Obd03, Obd07, BDHK06, Gan15, Sta23], where such lower bound results do not hold. For the most significant parameter—the number of colours in a Rabin game—we ask if algorithms can potentially benefit (both theoretically and practically) with some clever pre-processing. Modifying the input game graph or the acceptance condition to significantly reduce a suitable parameter or even finding a restriction of the trees necessary for lifting could lead to an improved performance of the algorithm. Reducing the size of such trees from more than $k!$ to a value closer to $(k/2)!$ in such cases is significant for practical algorithms. Indeed, 8! is comparable to the number of words in this thesis, while 4! is closer to the number of words in the following question.

Question VIII. *Can we solve Rabin games with k colours and n vertices in time that is polynomial in n and linear in $(k/c)!$ for some constant $c > 1$?*

Finally, we ask if we can extend our algorithms to solve games with a wider range of objectives. Toward this step, we considered fair Rabin games in Chapter 10. However, we can also ask the more general question of solving Muller games. It is already known that algorithms that solve Muller games by converting them to parity games that are fixed parameter tractable with respect to the number of colours. But, as with Rabin games, we face an exponential blow-up in space required to solve these games. Muller games are known to be PSPACE-complete [HD05] and algorithms that run in polynomial space are known [McN93].

Question IX. *Can we construct a polynomial-space algorithm that solves Muller games or Emerson-Lei games [EL86] in time that is* FPT *with respect to the number of colours?*

With these open avenues of thought, we hope to spur future investigations that will shape the trajectory of algorithms that solve parity and Rabin games faster in theory and practice.

Bibliography

[ANP21] André Arnold, Damian Niwiński, and Paweł Parys. A quasi-polynomial black-box algorithm for fixed point evaluation. In *CSL*, volume 183 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. (Cited on pages 7, 31, 31, 31, 32).

[BC17] Mikołaj Bojańczyk and Wojciech Czerwiński. *An Automata Toolbox*. Clemens Lode Verlag e.K., 2017. (Cited on pages 5, 6).

[BCJ18] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. (Cited on page 4).

[BDHK06] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. Dag-width and parity games. In *STACS*, pages 524–536, 2006. (Cited on pages 43, 219).

[BDM18] Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods Syst. Des.*, 52(2):193–226, 2018. (Cited on pages 8, 15, 102, 117, 117, 118, 123).

[BDM20] Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Robust worst cases for parity games algorithms. *Inf. Comput.*, 272:104501, 2020. (Cited on pages 118, 147).

[BDM+21] Massimo Benerecetti, Daniele Dell'Erba, Fabio Mogavero, Sven Schewe, and Dominik Wojtczak. Priority promotion with parysian flair. *CoRR*, abs/2105.01738, 2021. (Cited on pages 15, 117, 123).

[BG04] D. Berwanger and E. Grädel. Entanglement - A measure for the com-

plexity of directed graphs with applications to logic and games. In *LPAR*, pages 209–223, 2004. (Cited on page 43).

[BJP+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012. (Cited on page 149).

[BK10] Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *J. ACM*, 57(2):9:1–9:41, 2010. (Cited on page 209).

[BKMMP19] P. Baldan, B. König, C. Mika-Michalski, and T. Padoan. Fixpoint games on continuous lattices. *Proceedings of the ACM on Programming Languages*, 3(POPL, January 2019):26:1–26:29, 2019. (Cited on pages 2, 12, 31, 33, 33, 33).

[BL69] J. Richard Buchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *The Journal of Symbolic Logic*, 34(2):166–170, 1969. (Cited on page 3).

[BL19] Udi Boker and Karoliina Lehtinen. Register games. *CoRR*, abs/1902.10654, 2019. (Cited on page 2).

[BLV96] Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite games with streett and rabin chain winning conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 207–224, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. (Cited on page 8).

[BMM+22] Tamajit Banerjee, Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Sadegh Soudjani. A direct symbolic algorithm for solving stochastic Rabin games. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 81–98, Cham, 2022. Springer International Publishing. (Cited on pages 191, 192).

[BSV03] Henrik Björklund, Sven Sandberg, and Sergei Vorobyov. A discrete subexponential algorithm for parity games. In Helmut Alt and Michel Habib, editors, *STACS 2003*, pages 663–674, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on page 5).

[BT21] A. R. Balasubramanian and K. S. Thejaswini. Adaptive Synchronisation of Pushdown Automata. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory (CONCUR 2021)*, volume 203 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on page viii).

[BW18] J. C. Bradfield and I. Walukiewicz. *The mu-calculus and Model Checking*, pages 871–919. Springer, 2018. (Cited on pages 2, 2, 3).

[Bü62] J. Richard Büchi. On a decision method in restricted second order arithmetic. *International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11, 1962. (Cited on page 2).

[CAH11] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Qualitative concurrent parity games. *ACM Trans. Comput. Logic*, 12(4), July 2011. (Cited on pages 40, 40, 40, 41).

[CdAH05] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. The complexity of stochastic Rabin and Streett games'. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 878–890. Springer, 2005. (Cited on pages 192, 193, 193, 207, 207).

[CDF+19] W. Czerwiński, L. Daviaud, N. Fijalkow, M. Jurdziński, R. Lazić, and P. Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. (Cited on pages 6, 25, 43, 59, 60, 72, 74, 176, 176, 217).

[CDHS18] K. Chatterjee, W. Dvořák, M. Henzinger, and A. Svozil. Quasipolynomial set-based symbolic algorithms for parity games. In *LPAR-22*, volume 57 of *EPiC Series in Computing*, pages 233–253, Awassa, Ethiopia, 2018. EasyChair. (Cited on page 218).

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. (Cited on pages 2, 3).

[CF19] Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures*, pages 1–26, Cham, 2019. Springer International Publishing. (Cited on page 7).

[CFGO22] Thomas Colcombet, Nathanaël Fijalkow, Pawel Gawrychowski, and Pierre Ohlmann. The theory of universal graphs for infinite duration games. *Log. Methods Comput. Sci.*, 18(3), 2022. (Cited on pages 7, 167, 167).

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. (Cited on page 209).

[CHJ05] Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 178–187. IEEE Computer Society, 2005. (Cited on pages 194, 206).

[CHJ08] Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008. (Cited on page 191).

[CHP07] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International*

Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. (Cited on pages 208, 209).

[Chu57] Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. *Summaries of the Summer Institute of Symbolic Logic*, 1:3–50, 1957. (Cited on pages 3, 149).

[CJK+17] C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC 2017*, pages 252–263, Montreal, QC, Canada, 2017. ACM. (Cited on pages 5, 42, 46, 59).

[CJK+22] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasi-polynomial time. *SIAM Journal on Computing*, 51(2):STOC17–152–STOC17–188, 2022. (Cited on pages 4, 5, 8, 13, 17, 18, 18, 75, 117, 149, 150, 208, 212).

[Con92] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. (Cited on page 78).

[CPP+24] Antonio Casares, Marcin Pilipczuk, Michał Pilipczuk, Uéverton S. Souza, and K. S. Thejaswini. Simple and tight complexity lower bounds for solving rabin games. In *Symposium on Simplicity in Algorithms (SOSA)*, 2024. (Cited on page viii).

[CT20] Richard Combes and Mikael Touati. Solving random parity games in polynomial time. *CoRR*, abs/2007.08387, 2020. (Cited on page 216).

[dAF07] Luca de Alfaro and Marco Faella. An accelerated algorithm for 3-color parity games with an application to timed games. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, pages 108–120, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. (Cited on page 8).

[dAH00] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, LICS '00, page 141, USA, 2000. IEEE Computer Society. (Cited on page 40).

[DJL18] Laure Daviaud, Marcin Jurdziński, and Ranko Lazic. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 325–334. ACM, 2018. (Cited on pages 10, 22, 23, 44, 55, 79).

[DJL19] L. Daviaud, M. Jurdziński, and K. Lehtinen. Alternating weak automata from universal trees. In *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Amsterdam, the Netherlands, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on pages 2, 7, 10, 22, 44, 79).

[DJT20] Laure Daviaud, Marcin Jurdziński, and K. S. Thejaswini. The Strahler number of a parity game. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 123:1–123:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on pages vii, 16, 43, 79, 103, 117, 117, 181).

[DP11] Constantinos Daskalakis and Christos Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, page 790–804, USA, 2011. Society for Industrial and Applied Mathematics. (Cited on page 4).

[DS22] Daniele Dell'Erba and Sven Schewe. Smaller progress measures and separating automata for parity games. *Frontiers Comput. Sci.*, 4, 2022. (Cited on pages 6, 43, 102).

[EH86] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, jan 1986. (Cited on page 2).

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 328–337. IEEE Computer Society, 1988. (Cited on pages 2, 4, 7, 28, 150, 214, 214).

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. (Cited on pages 2, 2, 2, 2, 3, 3, 3, 4, 20, 20, 23, 27, 55, 55).

[EJ99] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 1999. (Cited on pages 3, 7, 18, 28, 150, 214, 214, 218).

[EJS93] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 1993. (Cited on pages 2, 3).

[EL86] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings of the Symposium on Logic in Computer Science (LICS '86), Cambridge, Massachusetts, USA, June 16-18, 1986*, pages 267–278. IEEE Computer Society, 1986. (Cited on pages 103, 219).

[ELS16] J. Esparza, M. Luttenberger, and M. Schlund. A brief history of Strahler numbers—with a preface. Technical report, Technical University of Munich, 2016. (Cited on page 42).

[Eri19] Leif Eriksson. Solving temporal CSPs via enumeration and SAT compilation. Master's thesis, Linköping University, 2019. (Cited on page 212).

[Ers58] A. P. Ershov. On programming of arithmetic operations. *Communications of the ACM*, 1(8):3–6, 1958. (Cited on page 42).

[EWS01] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 694–707. Springer, 2001. (Cited on page 2).

[Fea10] J. Fearnley. Exponential lower bounds for policy iteration. In *ICALP 2010*, volume 6199 of *LNCS*, pages 551–562, Bordeaux, France, 2010. Springer. (Cited on pages 14, 78).

[Fea17] John Fearnley. Efficient parallel strategy improvement for parity games. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 2017. (Cited on pages 8, 78).

[FGHS21] John Fearnley, Paul W. Goldberg, Alexandros Hollender, and Rahul Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, page 46–59, New York, NY, USA, 2021. Association for Computing Machinery. (Cited on page 4).

[FGO20] Nathanaël Fijalkow, Pawel Gawrychowski, and Pierre Ohlmann. Value iteration using universal graphs and the complexity of mean payoff games. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. (Cited on page 7).

[FHZ11] O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *STOC 2011*, pages 283–292, San Jose, CA, USA, 2011. ACM. (Cited on page 79).

[FJdK+19] J. Fearnley, S. Jain, B. de Keijzer, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *International Journal on Software Tools for Technology Transfer*, 21(3):325–349, 2019. (Cited on pages 5, 8, 16, 17, 42, 43, 102, 102, 117, 138, 150, 151).

[FK84] Nissim Francez and Dexter Kozen. Generalized fair termination. In *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, page 46–53, New York, NY, USA, 1984. Association for Computing Machinery. (Cited on pages 2, 149).

[FKL10] Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. (Cited on page 191).

[FL09] Oliver Friedmann and Martin Lange. Solving parity games in practice. In Zhiming Liu and Anders P. Ravn, editors, *Automated Technology for Verification and Analysis*, pages 182–196, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. (Cited on pages 8, 117).

[FL11] John Fearnley and Oded Lachish. Parity games on graphs with medium tree-width. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, pages 303–314, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on page 43).

[Fri09] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS 2009*, pages 145–156, Los Angeles, CA, USA, 2009. IEEE Computer Society. (Cited on pages 14, 78).

[Fri11] Oliver Friedmann. Recursive algorithm for parity games requires exponential time. *RAIRO Theor. Informatics Appl.*, 45(4):449–457, 2011. (Cited on pages 118, 128, 128, 129, 147).

[FS18] John Fearnley and Rahul Savani. The complexity of all-switches strategy improvement. *Log. Methods Comput. Sci.*, 14(4), 2018. (Cited on page 78).

[Gan15] Moses Ganardi. Parity games of bounded tree- and clique-width. In Andrew Pitts, editor, *Foundations of Software Science and Computation Structures*, pages 390–404, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. (Cited on pages 43, 219).

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982. (Cited on pages 5, 7, 8, 150).

[GI17] Hugo Gimbert and Rasmus Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. (Cited on page 5).

[GKK88] V.A. Gurvich, A.V. Karzanov, and L.G. Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988. (Cited on page 78).

[GL68] M. Kh. Gol'dberg and É. M. Livshits. On minimal universal trees. *Mathematical notes of the Academy of Sciences of the USSR*, 4:713–717, 1968. (Cited on page 176).

[GLM+15] Jakub Gajarský, Michael Lampis, Kazuhisa Makino, Valia Mitsou, and Sebastian Ordyniak. Parameterized algorithms for parity games. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2015. (Cited on page 218).

[GTW02] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. (Cited on page 2).

[GW13] Maciej Gazda and Tim A. C. Willemse. Zielonka's recursive algorithm: dull, weak and solitaire games and tighter bounds. In Gabriele Puppis and Tiziano Villa, editors, *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, Borca di Cadore, Dolomites, Italy, 29-31th August 2013*, volume 119 of *EPTCS*, pages 7–20, 2013. (Cited on page 118).

[HD05] Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In *30th International Symposium on Mathematical Foundations of Computer Science, MFCS 2005*, volume 3618 of *Lecture Notes in Computer Science*, pages 495–506. Springer, 2005. (Cited on page 219).

[HK66] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966. (Cited on page 78).

[HK07] P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. In *SODA*, pages 637–644, 2007. (Cited on page 43).

[HKLN12] Keijo Heljanko, Misa Keinänen, Martin Lange, and Ilkka Niemelä. Solving parity games by a reduction to sat. *Journal of Computer and System Sciences*, 78(2):430–440, 2012. Games in Verification. (Cited on page 8).

[Hor05] Florian Horn. Streett games on finite graphs. In *Games in Design and Verification*, 2005. (Cited on pages 8, 150).

[How60] R. A. Howard. *Dynamic Programming and Markov Processes. Pp. 136. 46s. 1960. (John Wiley and Sons, N.Y.).* The M.I.T Press, 1960. (Cited on pages 14, 78).

[HP22] Daniel Hausmann and Nir Piterman. A survey on satisfiability checking for the μ-calculus through tree automata. In Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Rupak Majumdar, editors, *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, volume 13660 of *Lecture Notes in Computer Science*, pages 228–251. Springer, 2022. (Cited on page 3).

[HS19] Daniel Hausmann and Lutz Schröder. Optimal satisfiability checking for arithmetic μ-calculi. In *Foundations of Software Science and Computation Structures*, pages 277–294. Springer International Publishing, 2019. (Cited on pages 2, 7, 31).

[HS21] Daniel Hausmann and Lutz Schröder. Quasipolynomial computation of nested fixpoints. In *TACAS*, volume 12651 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2021. (Cited on pages 31, 31, 31, 33).

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. (Cited on page 209).

[JL17] M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–9, Reykjavik, Iceland, 2017.

IEEE Computer Society. (Cited on pages 5, 8, 13, 16, 17, 17, 26, 42, 43, 46, 55, 55, 59, 59, 59, 60, 60, 65, 70, 74, 75, 75, 75, 75, 76, 79, 102, 103, 117, 117, 138, 138, 150, 150, 151, 151, 157, 172).

[JM20] M. Jurdziński and R. Morvan. A universal attractor decomposition algorithm for parity games. arXiv:2001.04333v1, 2020. (Cited on pages 7, 12, 26, 31, 119, 138).

[JMT22] Marcin Jurdziński, Rémi Morvan, and K. S. Thejaswini. Universal algorithms for parity games and nested fixpoints. In Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Rupak Majumdar, editors, *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, volume 13660 of *Lecture Notes in Computer Science*, pages 252–271. Springer, 2022. (Cited on pages vii, 7, 10, 15, 16, 26, 31, 32, 59, 59, 80, 102, 117, 117, 118, 119, 120, 138, 138, 218).

[JPA+22] Swen Jacobs, Guillermo A. Perez, Remco Abraham, Veronique Bruyere, Michael Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaetan Staquet, Clement Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021, 2022. (Cited on pages 8, 9).

[JPZ08] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008. (Cited on pages 5, 21, 120).

[Jur98] Marcin Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, nov 1998. (Cited on page 4).

[Jur00] M. Jurdziński. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301, Lille, France, 2000. Springer. (Cited on pages 5, 5, 8, 8, 55, 55, 59, 59, 74, 79, 102, 117, 127).

[Kei15] J. J. A. Keiren. Benchmarks for parity games. In *FSEN*, volume 9392 of *LNCS*, pages 127–142, Tehran, Iran, 2015. Springer. (Cited on page 117).

[KK91] N. Klarlund and D. Kozen. Rabin measures and their applications to fairness and automata theory. In *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 256–265, 1991. (Cited on pages 2, 17, 149, 151, 157, 157, 157, 164, 166).

[KKV01] Valerie King, Orna Kupferman, and Moshe Y. Vardi. On the complexity of parity word automata. In Furio Honsell and Marino Miculan, editors, *Foundations of Software Science and Computation Structures, 4th International Conference, FOSSACS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*, volume 2030 of *Lecture Notes in Computer Science*, pages 276–286. Springer, 2001. (Cited on pages 83, 84, 150).

[KL22] Zhuan Khye Koh and Georg Loho. Beyond value iteration for parity games: Strategy iteration with universal trees. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 63:1–63:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on pages 14, 79, 92).

[Kla90] Nils Klarlund. *Progress Measures and Finite Arguments for Infinite Computations*. PhD thesis, Cornell University, Ithaca, 1990. (Cited on page 157).

[KMSZ18] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 567–577, Cham, 2018. Springer International Publishing. (Cited on page 218).

[Knu73] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1973. (Cited on page 42).

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. Special Issue Ninth In-

ternational Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982. (Cited on page 2).

[KPV16] Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. (Cited on page 191).

[KV98] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Symposium on the Theory of Computing*, 1998. (Cited on pages 2, 7).

[LB20] Karoliina Lehtinen and Udi Boker. Register Games. *Logical Methods in Computer Science*, Volume 16, Issue 2, May 2020. (Cited on pages 6, 13, 42, 43, 46, 59, 75, 181, 216).

[LBC+94] David E. Long, Anca Browne, Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In David L. Dill, editor, *Computer Aided Verification*, pages 338–350, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg. (Cited on pages 2, 3, 5, 31, 103).

[LBD20] Ruben Lapauw, Maurice Bruynooghe, and Marc Denecker. Improving parity game solvers with justifications. In *Verification, Model Checking, and Abstract Interpretation*, page 449–470, Berlin, Heidelberg, 2020. Springer-Verlag. (Cited on pages 117, 123).

[Leh18] K. Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 639–648, Oxford, UK, 2018. IEEE. (Cited on pages 6, 43, 45, 45, 46, 46, 46, 46, 47, 47, 59, 76).

[LMS18] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly super-exponential parameterized problems. *SIAM Journal on Computing*, 47(3):675–702, 2018. (Cited on pages 8, 210, 210).

[LPSW22] Karoliina Lehtinen, Paweł Parys, Sven Schewe, and Dominik Wojtczak. A Recursive Approach to Solving Parity Games in Quasipolynomial Time. *Logical Methods in Computer Science*, Volume 18, Issue 1, January 2022. (Cited on pages 15, 16, 26, 27, 27, 80, 102, 117, 117, 117, 118, 138, 138, 138, 138).

[LSW19] K. Lehtinen, S. Schewe, and D. Wojtczak. Improving the complexity of Parys' recursive algorithm, 2019. (Cited on pages 7, 16, 27, 138, 138, 138).

[Lut08] Michael Luttenberger. Strategy iteration using non-deterministic strategies for solving parity games. *CoRR*, abs/0806.2923, 2008. (Cited on pages 14, 78).

[Mar75] Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. (Cited on page 20).

[McM93] Kenneth L. McMillan. *Mu-Calculus Model Checking*, pages 113–128. Springer US, Boston, MA, 1993. (Cited on page 3).

[McN66] Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. (Cited on pages 4, 149).

[McN93] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. (Cited on pages 4, 7, 10, 15, 15, 23, 28, 40, 80, 83, 102, 102, 103, 104, 117, 118, 219).

[MS95] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1):69–107, 1995. (Cited on pages 2, 3, 4).

[MSL18] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In *CAV(I)*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018. (Cited on pages 8, 9).

[MST23] Rupak Majumdar, Irmak Sağlam, and K. S. Thejaswini. Rabin games and colourful universal trees. unpublished, 2023. (Cited on page vii).

[Niw88] Damian Niwiński. Fixed points vs. infinite generation. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988*, pages 402–409. IEEE Computer Society, 1988. (Cited on page 2).

[Niw97] Damian Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189(1):1–69, 1997. (Cited on page 2).

[Obd03] J. Obdrzálek. Fast mu-calculus model checking when tree-width is bounded. In *CAV*, pages 80–92, 2003. (Cited on pages 43, 219).

[Obd07] J. Obdrzálek. Clique-width and parity games. In *CSL*, pages 54–68, 2007. (Cited on pages 43, 219).

[Ohl21] Pierre Ohlmann. *Monotonic graphs for Parity and Mean-Payoff games*. PhD thesis, ENS Paris-Saclay, 2021. (Cited on page 79).

[Ohl22] Pierre Ohlmann. Characterizing positionality in games of infinite duration over infinite graphs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. (Cited on page 7).

[Par19] P. Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In *MFCS 2019*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:13, Aachen, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on pages 7, 16, 26, 26, 27, 117, 118, 138, 138, 138).

[Par20] P. Parys. Parity games: Another view on Lehtinen's algorithm. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020*, volume 152 of *LIPIcs*, pages 32:1–32:15, Barcelona, Spain, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. (Cited on pages 6, 13, 43, 43, 45, 59).

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. (Cited on pages 2, 149).

[PP06] N. Piterman and A. Pnueli. Faster solutions of Rabin and Streett games. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 275–284, 2006. (Cited on pages 8, 150, 151, 172).

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In E. Allen Emerson and Kedar S. Namjoshi, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 364–380, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cited on page 4).

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, page 179–190, New York, NY, USA, 1989. Association for Computing Machinery. (Cited on pages 4, 7, 150).

[PT23] Aditya Prakash and K. S. Thejaswini. On history-deterministic one-counter nets. In Orna Kupferman and Pawel Sobocinski, editors, *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, volume 13992 of *Lecture Notes in Computer Science*, pages 218–239. Springer, 2023. (Cited on page viii).

[Pur95] Anuj Puri. *Theory of Hybrid Systems and Discrete Event Systems*. PhD thesis, EECS Department, University of California, Berkeley, Dec 1995. (Cited on page 78).

[QS83] J. P. Queille and J. Sifakis. Fairness and related properties in transition systems — a temporal logic to deal with fairness. *Acta Informatica*, 19(3):195–220, Jul 1983. (Cited on pages 3, 191, 191).

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969. (Cited on pages 2, 2, 27, 149).

[Saf88] Shmuel Safra. On the complexity of omega-automata. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 319–327. IEEE Computer Society, 1988. (Cited on pages 3, 4).

[Sch08] Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 369–384. Springer, 2008. (Cited on pages 14, 78).

[Sch17] S. Schewe. Solving parity games in big steps. *Journal of Computer and Systems Sciences*, 84:243–262, 2017. (Cited on page 5).

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, 1989. (Cited on page 3).

[Sei96] Helmut Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996. (Cited on pages 3, 5, 31, 32, 32).

[SF07] Sven Schewe and Bernd Finkbeiner. Distributed synthesis for alternating-time logics. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura, editors, *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, volume 4762 of *Lecture Notes in Computer Science*, pages 268–283. Springer, 2007. (Cited on page 5).

[Sha53] Lloyd S. Shapley. Stochastic games*. *Proceedings of the National Academy of Sciences*, 39:1095 – 1100, 1953. (Cited on page 40).

[SS23] Irmak Saglam and Anne-Kathrin Schmuck. Solving odd-fair parity games. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 34:1–34:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on page 207).

[Sta23] Konrad Staniszewski. Parity games of bounded tree-depth. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 33:1–33:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. (Cited on pages 43, 219).

[Sto74] Larry Joseph Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering., 1974. (Cited on page 4).

[Str81] Robert S. Streett. Propositional dynamic logic of looping and converse. In *Proceedings of the Thirteenth Annual ACM Symposium on*

Theory of Computing, STOC '81, page 375–383, New York, NY, USA, 1981. Association for Computing Machinery. (Cited on page 2).

[STSN24] Anne-Kathrin Schmuck, K. S. Thejaswini, Irmak Saglam, and Satya Prakash Nayak. Solving two-player games under progress assumptions. In Rayna Dimitrova, Ori Lahav, and Sebastian Wolff, editors, *Verification, Model Checking, and Abstract Interpretation - 25th International Conference, VMCAI 2024, London, United Kingdom, January 15-16, 2024, Proceedings, Part I*, volume 14499 of *Lecture Notes in Computer Science*, pages 208–231. Springer, 2024. (Cited on page viii).

[Tar55] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285 – 309, 1955. (Cited on page 168).

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. (Cited on page 83).

[The19] K. S. Thejaswini. *Parity Games and Register Index*. Master's thesis, Chennai Mathematical Institute, Chennai, India, 2019. (Cited on pages 13, 47).

[TOJ22] K. S. Thejaswini, Pierre Ohlmann, and Marcin Jurdzinski. A technique to speed up symmetric attractor-based algorithms for parity games. In Anuj Dawar and Venkatesan Guruswami, editors, *42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2022, December 18-20, 2022, IIT Madras, Chennai, India*, volume 250 of *LIPIcs*, pages 44:1–44:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. (Cited on page vii).

[Var96] Moshe Y. Vardi. Rank predicates vs. progress measures in concurrent-program verification. *Chic. J. Theor. Comput. Sci.*, 1996, 1996. (Cited on page 157).

[vD18] T. van Dijk. Oink: An implementation and evaluation of modern parity game solvers. In *Tools and Algorithms for the Construction and Analysis of Systems, 24th International Conference, TACAS 2018*, volume 10805 of *LNCS*, pages 291–308, Thessaloniki, Greece, 2018. Springer. (Cited on pages 9, 102, 117, 118, 123).

[vD19] Tom van Dijk. A parity game tale of two counters. In Jérôme Leroux and Jean-François Raskin, editors, *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, volume 305 of *EPTCS*, pages 107–122, 2019. (Cited on page 147).

[vdB10] Freark van der Berg. Solving parity games on the playstation 3. 2010. (Cited on page 8).

[Ven08] Yde Venema. Lecture notes on the modal μ-calculus, 2008. (Cited on page 12).

[Vie90] X. G. Viennot. Trees everywhere. In *15th Colloquium on Trees in Algebra and Programming*, volume 431 of *LNCS*, pages 18–41, Copenhagen, Denmark, 1990. Springer. (Cited on page 42).

[VJ00] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV 2000*, volume 1855 of *LNCS*, pages 202–215, Chicago, IL, USA, 2000. Springer. (Cited on pages 14, 78, 78).

[vW08] Jaco van de Pol and Michael Weber. A multi-core solver for parity games. *Electronic Notes in Theoretical Computer Science*, 220(2):19–34, 2008. Proceedings of the 7th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC 2008). (Cited on page 8).

[Wil01] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of The Belgian Mathematical Society-simon Stevin*, 8:359–391, 2001. (Cited on page 3, 3).

[YY90] Alexander Yakhnis and Vladimir Yakhnis. Extension of gurevich-harrington's restricted memory determinacy theorem: A criterion for the winning player and an explicit class of winning strategies. *Ann. Pure Appl. Log.*, 48(3):277–297, 1990. (Cited on page 5).

[Zie98] W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998. (Cited on pages 2, 2, 7, 10, 15, 15, 21, 22, 23, 23, 80, 83, 102, 102, 103, 104, 117, 117, 118, 119, 120, 218).

[ZP96] U. Zwick and M. Paterson. The complexity of mean-payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996. (Cited on page 78).