Dissertation / Doctoral Thesis

Titel der Dissertation / Title of the Doctoral Thesis

Improved Algorithms and Conditional Lower Bounds for Problems in Formal Verification and Reactive Synthesis

verfasst von / submitted by

Dipl. Ing. Veronika Loitzenbauer

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften (Dr. techn.)

Wien, 2016 / Vienna, 2016

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dr. Monika Henzinger

Abstract

Model checking is a fully automated approach in *formal verification* to either prove a system's correctness or find an error. It is an essential and widely-used component in the iterative design of systems such as microprocessors. In contrast to iterative design, *Church's synthesis problem* asks to automatically generate a correct system from its specification. *Reactive synthesis* is the synthesis of *reactive systems* that are systems that repeatedly interact with their environment.

For formal verification and synthesis mathematical models of systems and their behaviors are needed. *Directed graphs* are a fundamental model of systems. *Markov decision processes* (*MDPs*) additionally incorporate probabilistic behavior of, for example, randomized concurrent systems or communication protocols. A model for reactive systems are *game graphs*, where the vertices of the graph are partitioned between two players and one player represents controllable inputs and the other uncontrollable inputs. The *automata-theoretic* approach to model-checking and synthesis is a canonical way to formally specify the desired behaviors of a system using *ω-regular objectives* such as Büchi, parity, and Streett objectives. Additionally, *mean-payoff* objectives allow for expressing quantitative properties of systems such as resource consumption.

In this thesis we develop algorithms with *improved worst-case running times* for several problems on graphs, MDPs, and game graphs with ω-regular and mean-payoff objectives. Additionally, we show the first super-linear *conditional lower bounds* for polynomial-time problems in this area. In particular we present the following results:

- For mean-payoff objectives on graphs the first approximation algorithm that improves for dense graphs upon the long-standing running time bounds for exact algorithms.
- For Streett objectives the first sub-quadratic time algorithm for MDPs and a faster algorithm for dense MDPs and graphs.
- For parity games the first sub-cubic time algorithm for three priorities and improved symbolic algorithms for the general case.
- New algorithms and super-linear conditional lower bounds for conjunctions and disjunctions of basic ω-regular objectives. These results show for the first time that, under popular assumptions, there exist problems with strictly higher running times on MDPs than on graphs ("model separation") and that for each graph and MDPs there exist objectives with strictly higher running times compared to closely related objectives ("objective separation").
- For generalized Büchi games a new upper and tight conditional lower bounds that imply a model separation between MDPs and game graphs, and a faster algorithm for dense GR(1) games.

Zusammenfassung

Die *Modellprüfung* ist ein vollautomatisches Verfahren zur *formalen Verifikation*, die entweder die Korrektheit eines Systems zeigt oder einen Fehler findet. Sie ist ein essentieller und oft verwendeter Bestandteil im schrittweisen Design von Systemen, wie zum Beispiel von Mikroprozessoren. Im Gegensatz zu schrittweisem Design verlangt das *Syntheseproblem von Church* die automatische Generierung eines korrekten Systems aus einer vorgegebenen Spezifikation. *Reaktive Sythese* ist die Synthese von *reaktiven Systemen*, welche laufend mit ihrer Umgebung interagieren.

Für die formale Verifikation und Synthese werden mathematische Modelle von Systemen und ihrem Verhalten benötigt. *Gerichtete Graphen* sind ein grundlegendes Modell von Systemen. *Markow-Entscheidungsprozesse* (*MEPs*) können zusätzlich zufallsgesteuertes Verhalten abbilden, zum Beispiel von randomisierten parallelen Systemen und von Kommunikationsprotokollen. Ein Modell für reaktive Systeme sind *Spielgraphen*, bei denen die Knoten des Graphens zwischen einer Spielerin, die die kontrollierbaren Eingaben repräsentiert, und ihrem Gegenspieler, der die unkontrollierbaren Eingaben repräsentiert, aufgeteilt sind. Der *Automaten-basierte* Ansatz zur Modellprüfung und Synthese ist eine anerkannte Methode um das erwünschte Verhalten von Systemen mit Hilfe von *ω-regulären Zielvorgaben* wie Büchi-, Paritäts- oder Streett-Zielvorgaben formal zu beschreiben. Zusätzlich können quantitative Eigenschaften wie Ressourcenverbrauch durch *Mittelwerts-Zielvorgaben* ausgedrückt werden.

In dieser Arbeit entwickeln wir Algorithmen mit *verbesserter Laufzeit* für mehrere Probleme auf Graphen, MEPs, und Spielgraphen mit ω-regulären Zielvorgaben und Mittelwerts-Zielvorgaben. Zusätzlich zeigen wir die ersten super-linearen *bedingten unteren Schranken* für Polynomialzeitprobleme in diesem Gebiet. Konkret präsentieren wir die folgenden Ergebnisse:

- Für Mittelwerts-Zielvorgaben auf Graphen den ersten Approximationsalgorithmus, der für dichte Graphen die lange bekannten Laufzeitschranken für exakte Algorithmen durchbricht.
- Für Streett-Zielvorgaben den ersten Algorithmus mit weniger als quadratischer Laufzeit sowie verbesserte Algorithmen für dichte MEPs und Graphen.
- Für Paritätsspiele den ersten sub-kubischen Algorithmus für drei Prioritäten sowie verbesserte symbolische Algorithmen für den allgemeinen Fall.
- Neue Algorithmen und super-lineare bedingte untere Schranken für Konjunktionen und Disjunktionen von einfachen ω-regulären

Zielvorgaben. Diese Ergebnisse zeigen zum ersten Mal, dass es unter weitverbreiteten Annahmen für MEPs strikt höhere Laufzeitschranken als für Graphen ("Modell-Separierung") und für manche Zielvorgaben strikt höhere Laufzeitschranken als für nah verwandte Zielvorgaben ("Zielvorgaben-Separierung") gibt.

- Für verallgemeinerte Büchi Spiele einen neuen Algorithmus und passende bedingte untere Schranken, die eine Modellseparierung zwischen MEPs und Spielgraphen implizieren, sowie für GR(1) Spiele einen schnelleren Algorithmus auf dichten Graphen.

Acknowledgments

First and foremost I would like to thank my advisor. Monika, you have done more for me and my personal and professional development than you can possibly imagine. Having you as a mentor and role model has been incredibly important to me. Without you, I wouldn't have discovered my favorite profession and wouldn't have had the courage to follow my path. Thank you for everything!

This thesis would not have been possible without Krish. Thank you for your superb guidance in all our collaborations! I'm indebted to Wolfgang, Sebastian, and Oren for the tons of advice and support. Thank you for the great collaborations and for being there for me. Stefan and Gramoz, thank you for sharing and sticking together, you made the last year of my PhD the best one.

Special thanks go to Jean-François and Piotr for serving on my thesis committee, and to everyone who proofread or reviewed parts of this thesis for greatly improving its presentation.

I'm grateful to Pino and Seth for the collaborations and for hosting and supporting me. I would like to thank Lara, Hana, and Christina for being my allies; all the other colleagues and visitors at TAA for the nice atmosphere in Vienna; Nikos, Eugeniya, and Frederik for my wonderful time in Rome; Shiri, Thomas, Sayan, and Slobo for the interesting collaborations; Valerie, Tsvi, and Sven for their encouragement; and Oded Goldreich for sharing his advice to himself[1]: *"The personal and professional are tightly linked it is."*

I would like to take this opportunity to deeply thank my family, including my grandparents and my great-grandmother, for their unconditional support and believe in me. To my parents, their partners, and my siblings: you are simply awesome. I couldn't imagine a better family.

I'm extremely grateful to all the amazing people I've connected with along the journey of my PhD. Take care and be brave! From the bottom of my heart, I would like to thank Manuel, Sharon, Lea, Becci, Ricky, Julia, and Nicole for being there when I needed it most; Martin, Dominique, Heinz & Bernd, Elisabeth J., Reinhard, Phebe, Bettina, and Jan for inspiration and support in the right moments; Christian for our deep connection; Chris for the trust, the playfulness, and all that you've taught me; and Gaya and Alma for being awesome ☺

Last but not least I would like to thank Brené Brown and Amanda Palmer for standing in the arena with all their vulnerability. Your names stand as an example for creating a world in which we can be ourselves without fear.

[1] `http://www.wisdom.weizmann.ac.il/~oded/advice.html`.

Bibliographic Note

The chapters of this thesis are based on the following publications and manuscripts:

- **Chapter 3:** Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger, Veronika Loitzenbauer, and Michael A. Raskin. "Approximating the minimum cycle mean". In: *Theoretical Computer Science* 547 (2014), pp. 104–116. DOI: 10.1016/j.tcs.2014.06.031. Announced at GandALF'13.
- **Chapter 4 and 5:** Krishnendu Chatterjee, Monika Henzinger, and Veronika Loitzenbauer. "Improved Algorithms for One-Pair and k-Pair Streett Objectives". In: *Symposium on Logic in Computer Science (LICS).* 2015, pp. 269–280. DOI: 10.1109/LICS.2015.34. Full version available at http://arxiv.org/abs/1410.0833.
- **Chapter 5:** Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Improved Set-Based Symbolic Algorithms for Parity Games". 2016. Unpublished.
- **Chapter 4 and 6:** Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Model and Objective Separation with Conditional Lower Bounds: Disjunction is Harder than Conjunction". In: *Symposium on Logic in Computer Science (LICS).* 2016, pp. 197–206. DOI: 10.1145/2933575.2935304. Full version available at http://arxiv.org/abs/1602.02670.
- **Chapter 7:** Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Conditionally Optimal Algorithms for Generalized Büchi Games". In: *Symposium on Mathematical Foundations of Computer Science (MFCS).* 2016, 25:1–25:15. DOI: 10.4230/LIPIcs.MFCS.2016.25. Full version available at http://arxiv.org/abs/1607.05850.

The research leading to these results has received funding from the Vienna Science and Technology Fund (WWTF) grants ICT10-002 and ICT15-003, the Austrian Science Fund (FWF): P23499-N23, the Austrian Marshall Plan Foundation, and the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

Contents

CHAPTER 1

Introduction

Errors in software and hardware design may endanger our safety and incur huge costs, such as in the control of an aircraft or in the design of a CPU. Therefore, provably error-free systems would be highly desirable. Such a system can be anything from a traffic light or a vending machine to a multi-threaded program, a communication process, or a hardware circuit [BK08, Ch. 1–2]. Proving the correctness of systems is unachievable in general since even determining whether an arbitrary given program terminates is undecidable [see e.g. Sip06, Ch. 4.2]. However, in the last decades many useful formal methods have been developed by focusing on relevant special cases, and many practical tools are widely available [Hol97; KNP11]. *Formal verification* is an essential component in the iterative design of systems such as microprocessors, communication protocols, and security algorithms [see e.g. Hol93; CW96]. *Model checking* is a fully automated approach in verification to either prove a system's correctness or provide a counter example for it. Its major role in todays' software and hardware industry is highlighted by the Turing award in 2007 for the founders of model checking [CE81; QS82] for transforming technology with their theoretical research [Com08]. While verification is mostly used in iterative design processes of systems, the *synthesis problem* [Chu62; Büc62] asks to automatically generate a correct system from its specification. *Reactive synthesis* is the synthesis of *reactive systems* that are systems that repeatedly interact with their environment.

For both verification and synthesis we need mathematical *models* of systems and a formal way to express a system's *specification*, that is, the desired behaviors of the system. The fundamental models of systems can be represented as *directed graphs*. *Markov decision processes* (*MDPs*) additionally incorporate probabilistic behavior of, e.g., randomized concurrent systems [Var85; CY95] or communication protocols [Boh+03]. Reactive systems are modeled as *game graphs* [BL69; McN93; Tho95], where the vertices of the graph are partitioned between two players and one player represents the system and controllable environment inputs and its opponent represents uncontrollable environment inputs. The *automata-theoretic* approach [VW86; Var96] to model-checking and synthesis is a canonical way to formally specify the desired behaviors of a system using *ω-regular objectives* such as Büchi, parity, and Streett objectives. Additionally, *mean-payoff objectives* allow for expressing quantitative properties of systems such as resource consumption.

In this thesis we consider algorithmic questions emerging from model checking and synthesis. In the remainder of this chapter we first describe the relevant models and specifications in more detail and then provide an overview over the algorithmic problems, related work, and our results.

Models. A model of a system typically focuses on the control and interaction between different parts of the system, or the interaction between the system and its environment, and ignores implementation details.

Finite directed graphs are a model for non-deterministic systems, and provide the framework to model hardware and software systems [Hol97; Cim+00] as well as many basic logic-related questions such as automata emptiness. The vertices of the graph represent the states of the system and the edges represent the transitions between states. Multiple outgoing edges of a vertex represent several possible behaviors of the system emerging from, e.g., nondeterminism in the parallel or distributed execution of processes [Kel76] or, during the iterative design of a system, from different design choices [see e.g. BK08, Ch. 2].

Markov decision processes (*MDPs*) model systems with both non-deterministic and probabilistic behavior. They are graphs in which a subset of vertices has a probability distribution over their outgoing edges. These probabilistic choices can express, e.g., randomization in concurrent systems [Var85; CY95], randomized communication protocols [Boh+03], lossy communication channels, or experimental data about the environment of the system [see e.g. BK08, Ch. 10].

In *game graphs* the vertices are partitioned between *two players*. Typically one player represents the system and the other the environment [AHK02] or, as in the verification of branching-time properties of reactive systems [EJ91], one player models the existential quantifiers and the opponent models the universal quantifiers. Two-player games on graphs are useful in many problems in verification and synthesis of systems such as the synthesis of reactive systems [Chu62; PR89; RW87], verification of open systems [AHK02], and checking interface compatibility [AH01] and well-formedness of specifications [Dil89]. Moreover, game-theoretic formulations have been used for program repair [JGB05] and the synthesis of programs [Cer+11].

Specifications. In this work we mostly consider ω-regular *objectives* [Tho97] (ω-regular languages extend regular languages to infinite words), which can express most commonly used properties in formal verification and reactive synthesis [MP92]. We consider for each objective also its complement or *dual* because an objective can represent either desired or undesired behaviors, and in games the two players have complementary objectives.

The most basic properties of safety-critical systems are *safety* objectives, that is, we want to verify that a given set of bad states can be avoided. An example for such a property is whether in a parallel system two processes can ever be in a critical section at the same time. The dual objective to safety is *reachability*, that is, a set of good states is reached eventually.

Another type of property is, e.g., whether every request of a process to enter a critical section is eventually granted. This corresponds to *liveness* or *Büchi* objectives, where a set of good states has to be reached *infinitely often*. Its dual objective is the *co-Büchi* objective, where a set of bad states may be reached only finitely often.

Streett objectives and their dual *Rabin* objectives can express all ω-regular languages [Saf88; Tho97]. Streett objectives directly correspond to *strong fairness* conditions; a scheduler, e.g., is *strongly fair* if every event that is enabled infinitely often is scheduled infinitely often. Büchi and co-Büchi objectives are important special cases of Streett and Rabin objectives.

Parity objectives generalize Büchi objectives and are a special case of both Streett and Rabin objectives [Cha07]. The dual of a parity objective is again a parity objective. Streett and Rabin objectives can be converted to parity objectives [Saf92] (with an exponential blow-up in the size of the model). Parity objectives are particularly important as they are equivalent to modal μ-calculus [EJ91], one of the most important logics in model-checking.

A different generalization of parity objectives that goes beyond ω-regular objectives are *mean-payoff* objectives. Here the edges are assigned weights and the objective is specified in terms of the average weight over a sequence of edges. In contrast to the other objectives, mean-payoff objectives are *quantitative objectives* that can model, e.g., the average resource consumption or delay of a system.

Algorithmic questions. To reason about all possible executions of a system, we consider *traces* or *plays* that are infinite paths induced by moving a token indefinitely along the edges of the model. The resolution of the nondeterministic choices at a vertex, that is, the choice of an outgoing edge at a non-random vertex, is called a *strategy*.

In *graphs* we ask, for each start vertex, whether there exists a trace that satisfies the objective. The set of start vertices for which this is the case is called the *winning set*. The algorithmic question for graphs is to determine the winning set and possibly strategies for vertices in the winning set.

In *MDPs* we also have random vertices and at a random vertex the outgoing edge is chosen according to the given probability distribution. In MDPs we ask, for each start vertex, whether there exists a strategy (for the non-random vertices) such that the objective is satisfied with probability 1. The set of start vertices for which this is satisfied is called the *almost-sure winning set*. The requirement to satisfy the objective with probability 1 is often called *qualitative* analysis of MDPs and it is common in the analysis of randomized distributed algorithms [KNP00; PSL00;

Sto03]. For a *quantitative* analysis of MDPs that determines the value of the winning probability from each start vertex see, e.g., [CH07].

In *game graphs* the plays are induced by the two players taking turns in moving the token along the edges of the graph. A *strategy* for a player is a function that describes for each vertex how the player that owns the vertex chooses to move tokens along the outgoing edges of the vertex to extend plays, and a *winning strategy* ensures the desired set of plays against all strategies of the opponent. The *winning set* of a player is the set of all start vertices for which she has a winning strategy. For all considered games the winning sets of the two players form a partition of the graph [Mar75]. The algorithmic question is to determine this partition and, possibly, to construct winning strategies for both players.

Symbolic model-checking. A fundamental difficulty in the model-checking approach is the size of the models as the number of states of a system is exponential in the number of variables used to describe the system. This is called the *state-space explosion problem.* One approach to deal with this difficulty is the use of *symbolic* algorithms, where the states and transitions of a system are not constructed or stored explicitly; instead, a specific set of operations can be applied to *sets* of states. A set can be encoded, e.g., with a binary decision diagram (BDD) [Lee59; Jr78; Bry86]. For game graphs with parity objectives, or *parity games* for short, we also consider symbolic algorithms.

Conditional lower bounds. In this thesis we mostly study problems for which polynomial-time algorithms are known, parity games being the only exception. In complexity theory, polynomial-time algorithms are seen as efficient. However, for huge graphs as in model checking, the difference between a quadratic-time and a linear-time algorithm can affect whether an automated procedure is feasible. Therefore, we would like to find faster algorithms also for polynomial-time problems, or show that no such algorithms can exist. For algorithmic problems unconditional super-linear lower bounds are very rare when polynomial-time (but super-linear) solutions exist. However, recently there have been many interesting results that establish *conditional lower bounds* for various combinatorial problems, see [Vas15] for a survey. These are lower bounds based on the assumption that for some well-studied problem no improvement over the best-known running time is possible (apart from lower order terms). Problems for which such conjectures were made include Boolean matrix multiplication [AN96; Lee02; Kav12; Hen+15], CNF-SAT [IPZ01; PW10; AV14; Car+16; Abb+16], 3-SUM [Pat10; GO12; AL13; KPP16], all-pairs shortest paths [RZ11; VW13], and problems related to cliques in graphs [Woe04; VW10; JV13; AVY15; ABV15a]. The lower bounds in this work assume that (A1) there is no combinatorial algorithm for multiplying two $n \times n$ Boolean matrices with a running time of $O(n^{3-\varepsilon})$ for any $\varepsilon > 0$; or (A2) for all $\varepsilon > 0$ there exists a k such that there is no algorithm for the k-CNF-SAT problem that runs in $2^{(1-\varepsilon)\cdot n} \cdot \text{poly}(m)$ time, where n is the number of variables and m the number of clauses. Combinatorial here means avoiding fast matrix multiplication [Le 14], see also the discussion in [Hen+15; Yu15; LW17]. Assumption (A2) is known as the Strong Exponential Time Hypothesis (SETH). These two assumptions have been used to establish lower bounds for several well-studied problems, such as dynamic graph algorithms [AV14; AVY15], measuring the similarity of strings [AVW14; Bri14; BK15; BI15; ABV15b], context-free grammar parsing [Lee02; ABV15a], and verifying first-order graph properties [PW10; Wil14b]. To the best of our knowledge, no relation between conjectures (A1) and (A2) is known.

For several basic model-checking questions the best-known upper bounds are quadratic or cubic and no super-linear lower bounds are known. In this thesis we present several algorithms with improved running times, including the first sub-quadratic time algorithm for MDPs with Streett objectives and the first sub-cubic time algorithm for parity-3 games. Furthermore, we establish the first conditional lower bounds that are super-linear for fundamental polynomial-time problems in model-checking and synthesis.

1.1 Related Work

In this section we present an overview of the algorithmic results for the relevant models and objectives. Let n denote the number of vertices and m the number of edges of the input model. For parity objectives another important input parameter is the number of *priorities* c and for mean-payoff objectives W denotes the maximum weight of an edge. We simplify the running times and omit dependencies on other input parameters for the sake of readability; precise running time bounds and more related work can be found in the subsequent chapters of the thesis.

For *graphs* the winning set for reachability, safety, Büchi, and co-Büchi objectives can be determined in linear time [Tar72] and for parity objectives in $O(m \log n)$ time [CH11]. For Streett objectives there is an $O(m \min(\sqrt{m \log n}, n))$ time algorithm [HT96]. The trivial algorithm for Rabin objectives takes time $O(mn)$. For mean-payoff objectives the best known algorithms run in time $O(mn)$ [Kar78] and $O(m\sqrt{n} \log(nW))$ [OA92].

For *MDPs* linear-time algorithms are only known for safety objectives (for which the problem is equivalent to the one in game graphs). The almost-sure winning set for reachability, Büchi, and co-Büchi objectives can be computed in $O(\min(m^{1.5}, n^2))$ time [CJH03; CH14] and with an additional factor of $\log n$ also for parity objectives [CH11]. For Streett and Rabin objectives running times of $O(n \cdot \min(m^{1.5}, n^2))$ follow from [CH14]. Mean-payoff objectives on MDPs can be solved in polynomial time by linear programming and in pseudo-polynomial time $O(mnW)$ [FV97].

For *game graphs* the running time for an objective and its dual is the same since the two objectives correspond to the objectives of the two players and one winning set is the complement of the other [Mar75]. For reachability and safety objectives the winning set can be computed in $O(m)$ time [Bee80; Imm81]. For Büchi and co-Büchi objectives, the current best known algorithm requires $O(n^2)$

time [CH14]. For Streett and Rabin objectives, the problem is coNP-complete and NP-complete [EJ88], respectively, and for one-pair Streett and one-pair Rabin objectives there is an $O(mn)$ time algorithm [Jur00; Sch08]. Parity games, and their generalization mean-payoff games, are one of the rare "natural" problems in $NP \cap coNP$ for which no polynomial-time algorithm is known. The best known algorithms for parity games run in time $n^{O(\sqrt{n})}$ [JPZ08] and (roughly) time $O(m \cdot n^{c/3})$ [Sch07]. Büchi games are parity games with $c = 2$; parity games with $c = 3$ are equivalent to one-pair Streett objectives. The best known algorithms for mean-payoff games run in pseudo-polynomial time $O(mnW)$ [Bri+11] and randomized sub-exponential time $O(2^{\sqrt{n \log n}} \log W)$ [BV07].

Symbolic algorithms for parity games. Parity games can be solved with $O(n^c)$ symbolic steps when storing a linear number of sets [EL86; Zie98] or with $O(n^{c/2+1})$ symbolic steps when storing $O(n^{c/2+1})$ many sets [Bro+97; Sei96].

1.2 Results and Outline

We survey the results of this thesis. The running times stated here are simplified; the actual improvements also depend on other input parameters. All algorithms for ω-regular objectives return (almost-sure) winning sets and can be modified to additionally construct winning strategies within the same time bounds. More details can be found in the corresponding chapters.

- In **Chapter 2** we provide definitions and preliminaries.
- In **Chapter 3** we present the first *approximation* algorithm for *mean-payoff objectives on graphs* that uses fast matrix multiplication to improve the dependence of the running time on n compared to the long-standing running time bounds for exact algorithms.
- In **Chapter 4** we present improved algorithms for *Streett objectives and MDPs.* In their simplified form, the running times are $O(m^{1.5}\sqrt{\log n})$ and $O(n^2)$, which removes a factor of $n/\sqrt{\log n}$ resp. n for MDPs and improves the running time for graphs when $m \in \omega(n^{4/3}/\sqrt[3]{\log n})$.
- In **Chapter 5** we first present an improved algorithm for *parity games* with a constant number of priorities and $m \in \omega(n^{4/3})$. Second, we show a *symbolic* algorithm that takes $O(n^{c/3+1})$ symbolic steps using a linear number of sets and therefore improves the number of symbolic steps of the best known symbolic algorithm while using the same number of sets as the basic symbolic algorithm.
- In **Chapter 6** we present several new algorithms and conditional lower bounds for Rabin objectives and disjunctions of reachability, safety, Büchi, and co-Büchi objectives on graphs and MDPs. These results lead to (1) a *separation of models*, that is, we provide conditional lower bounds for algorithmic

problems on MDPs and strictly lower upper bounds for the corresponding problems on graphs and (2) a *separation of objectives*, that is, for the same model conditional lower bounds for one objective that are strictly higher than the best upper bound for a related objective. In particular, together with the algorithms in Chapter 4, we separate Streett and Rabin objectives on both graphs and MDPs with respect to their asymptotic running times.

- *Generalized Büchi* objectives are conjunctions of Büchi objectives, and *GR(1)* objectives are implications of generalized Büchi objectives. In **Chapter 7** we provide a new algorithm and tight conditional lower bounds for generalized Büchi games that also imply a model separation between MDPs and game graphs. We further present an algorithm for GR(1) objectives that achieves an improved running time when $m \in \omega(n^{1.5})$.

CHAPTER 2

Preliminaries

In this chapter we first state the considered algorithmic questions more precisely by formally defining models, objectives, strategies, and winning sets. We then introduce basic algorithmic concepts that are repeatedly used in Chapters 4 to 7. Finally, we define fine-grained reductions and formally state the conjectures that we use for the conditional lower bounds in Chapters 6 and 7.

2.1 Models

Graphs. The simplest model of a system is a *directed graph*, where vertices correspond to states of the system and edges to system transitions. Unless stated otherwise, we consider finite directed graphs that might have self-loops but have at most one edge between every pair of vertices. We denote a graph by $G = (V, E)$, where V is its set of vertices and $E \subseteq V \times V$ is its set of edges. We refer to sets of vertices as vertices and to sets of edges as edges for short. We denote the number of vertices by $n = |V|$ and the number of edges by $m = |E|$. We assume that every vertex has at least one incoming edge and one outgoing edge and therefore we have $m \geq n$. Let $Out(G, u) = \{v \in V \mid (u, v) \in E\}$ be the set of successor vertices of vertex u in the graph G, let $Outdeg(G, u) = |Out(G, u)|$ be the number of successor vertices of vertex u in the graph G, and, analogously for the predecessor vertices, let $In(G, u) = \{v \in V \mid (v, u) \in E\}$ and $Indeg(G, u) = |In(G, u)|$; we omit G if clear from the context. The *reverse graph* $RevG = (V, E^R)$ of a graph $G = (V, E)$ is the graph with vertices V and all edges of E reversed, i.e., $E^R = \{(u, v) \mid (v, u) \in E\}$.

Markov decision processes (MDPs). An *MDP* $P = ((V, E), (V_1, V_R), \delta)$ consists of a finite directed graph $G = (V, E)$ with a set of vertices V and a set of edges E and a partition of the vertices V into *player-1 vertices* V_1 and *random vertices* V_R, and a *probabilistic transition function* δ. We call an edge (u, v) with $u \in V_1$ *player-1 edge* and an edge (w, v) with $w \in V_R$ a *random edge*. The probabilistic transition function δ is a function from V_R to $\mathcal{D}(V)$, where $\mathcal{D}(V)$ is the set of probability distributions over V. A random edge (v, w) is in E if and only if $\delta(v)[w] > 0$. We assume for simplicity that for each random vertex v the probability distribution $\delta(v)$ is the uniform distribution over all $w \in V$ with $(v, w) \in E$; this is w.l.o.g. in the qualitative analysis of MDPs. If $V_R = \emptyset$, then we have a graph, and if $V_1 = \emptyset$, then we have a Markov chain. Thus MDPs generalize graphs and Markov chains.

Game graphs. A *game graph* $\mathcal{G} = ((V, E), (V_1, V_2))$ is a finite directed graph $G = (V, E)$ with a set of vertices V and a set of edges E and a partition of V into *player-1 vertices* V_1 and *player-2 vertices* V_2. Given such a game graph $\mathcal{G}$, we denote with $\overline{\mathcal{G}}$ the game graph where the player-1 and player-2 vertices of $\mathcal{G}$ are interchanged, i.e, $\overline{\mathcal{G}} = ((V, E), (V_2, V_1))$. We use z to denote a player and $\overline{z}$ to denote her opponent. Graphs are a special case of game graphs with $V_2 = \emptyset$.

2.2 Plays and Strategies

Plays. An *infinite path* or *play* in a graph $G = (V, E)$ is an infinite sequence of vertices $\omega = \langle v_0, v_1, v_2, \dots \rangle$ such that $(v_\ell, v_{\ell+1}) \in E$ for all $\ell \geq 0$. We denote by Ω the set of all plays. In game graphs, plays are formed by the following *alternating game* of the two players: A game is initialized by placing a token on a vertex. Then the two players form an infinite path in the game graph by moving the token along the edges of the underlying graph. Whenever the token is on a vertex of V_z, player z moves the token along one of the outgoing edges of the vertex. In MDPs we have a random player instead of the second player and whenever the token is at a vertex v of V_R, the next vertex is chosen from $Out(v)$ according to the probability distribution $\delta(v)$.

Strategies. Let $V^* \cdot V_z$ denote the set of finite sequences of vertices of V that end in V_z. A *strategy* $\sigma : V^* \cdot V_z \to V$ of a player z is a function that given a finite prefix $\omega \in V^* \cdot V_z$ of a play ending at $v \in V_z$ selects a vertex $\sigma(\omega) \in Out(v)$ to extend the play. We denote by Σ the set of all player-1 strategies and by Π the set of all player-2 strategies. In game graphs a start vertex v together with a strategy $\sigma \in \Sigma$ for player 1 and a strategy $\pi \in \Pi$ for player 2 describe a unique play $\omega(v, \sigma, \pi) = \langle v_0, v_1, v_2, \dots \rangle$, which is defined as follows: $v_0 = v$ and for all $\ell \geq 0$, if $v_\ell \in V_1$, then $\sigma(\langle v_0, \dots, v_\ell \rangle) = v_{\ell+1}$, and if $v_\ell \in V_2$, then $\pi(\langle v_0, \dots, v_\ell \rangle) = v_{\ell+1}$. *Memoryless strategies* do not depend on the history of a play but only on the current vertex, i.e., we have $\sigma(\omega) = \sigma(\omega')$ for any $\omega, \omega' \in V^* \cdot V_z$ that end in the same vertex $v \in V_z$.

2.3 Objectives

A player-z *objective* ϕ is a subset of Ω said to be winning for player z. We say that a play $\omega \in \Omega$ *satisfies the objective* if $\omega \in \phi$. Let $\text{Inf}(\omega)$ for $\omega \in \Omega$ denote the set of vertices that occurs infinitely often in ω. Objectives for which the set of desired plays is determined by the set of vertices $\mathrm{Inf}(\omega)$ for $\omega \in \Omega$ are called *prefix-independent*. In game graphs we consider zero-sum games, where for a player-1 objective ϕ the dual objective $\Omega \setminus \phi$ is winning for player 2. We denote a game $(\mathcal{G}, \phi)$ by its game graph $\mathcal{G}$ and the player-1 objective ϕ. Below we define the objectives used in this work.

Reachability and safety objectives. For a set of *target vertices* $T \subseteq V$ the *reachability* objective is the set of infinite paths that contain a vertex of T, i.e., $\mathrm{Reach}(T) = \{\langle v_0, v_1, v_2, \dots \rangle \in \Omega \mid \exists i \geq 0 : v_i \in T\}$. Its dual objective is the *safety* objective, which for a vertex set $T \subseteq V$ is the set of infinite paths that *do not contain* any vertex of T, i.e., $\mathrm{Safe}(T) = \{\langle v_0, v_1, v_2, \dots \rangle \in \Omega \mid \forall i \geq 0 : v_i \notin T\}$. We have $\mathrm{Reach}(T) = \Omega \setminus \mathrm{Safe}(T)$.

Büchi and co-Büchi objectives. For a set of target vertices $T \subseteq V$ the *Büchi* objective is the set of infinite paths in which a vertex of T occurs *infinitely often*, i.e., $\text{Büchi}(T) = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T \neq \emptyset\}$. Its dual objective is the *co-Büchi* objective, which for a vertex set $T \subseteq V$ is the set of infinite paths for which *all* vertices of T occur only *finitely often*, i.e., $\text{coBüchi}(T) = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T = \emptyset\}$. We have $\text{Büchi}(T) = \Omega \setminus \text{coBüchi}(T)$.

Rabin and Streett objectives. We are given a set TP of k pairs (L_j, U_j) of vertex sets $L_j, U_j \subseteq V$ with $1 \leq j \leq k$, also called *target pairs*. The *Streett* objective is the set of infinite paths for which it holds *for each* $1 \leq j \leq k$ that whenever a vertex of L_j occurs infinitely often, then a vertex of U_j occurs infinitely often, i.e., $\mathrm{Streett}(\mathrm{TP}) = \{\omega \in \Omega \mid L_j \cap \mathrm{Inf}(\omega) = \emptyset \text{ or } U_j \cap \mathrm{Inf}(\omega) \neq \emptyset \text{ for all } 1 \leq j \leq k\}$. Its dual objective is the *Rabin objective*. The Rabin objective is the set of infinite paths for which there *exists* a j, $1 \leq j \leq k$, such that a vertex of L_j occurs infinitely often but no vertex of U_j occurs infinitely often, i.e., $\mathrm{Rabin}(\mathrm{TP}) = \{\omega \in \Omega \mid L_j \cap \mathrm{Inf}(\omega) \neq \emptyset \text{ and } U_j \cap \mathrm{Inf}(\omega) = \emptyset \text{ for some } 1 \leq j \leq k\}$. We have $\mathrm{Streett}(\mathrm{TP}) = \Omega \setminus \mathrm{Rabin}(\mathrm{TP})$. We denote the total number of elements for a set $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$ of target pairs with $b = \sum_{j=1}^{k}(|L_j| + |U_j|)$.

Conjunctive and disjunctive objectives. In general for two objectives ϕ_1 and ϕ_2 their conjunctive objective $\phi_1 \wedge \phi_2$ is equal to $\phi_1 \cap \phi_2$ and their disjunctive objective $\phi_1 \vee \phi_2$ is equal to $\phi_1 \cup \phi_2$, i.e., the conjunctive objective is satisfied when both objectives are satisfied and the disjunctive objective is satisfied when at least one of the two objectives is satisfied. For k objectives $\phi_1, \dots, \phi_k$ we denote their conjunctive objective with $\bigwedge_{j=j}^{k} \phi_j$ and their disjunctive objective with $\bigvee_{j=1}^{k} \phi_j$. In this notation we have $\mathrm{Streett}(\mathrm{TP}) = \bigwedge_{j=1}^{k} \left(\text{coBüchi}(L_j) \vee \text{Büchi}(U_j)\right)$ and $\mathrm{Rabin}(\mathrm{TP}) = \bigvee_{j=1}^{k} \left(\text{Büchi}(L_j) \wedge \text{coBüchi}(U_j)\right)$.

A *generalized (or conjunctive) Büchi objective* is specified by a set of k target sets T_j for $1 \leq j \leq k$ and is equal to $\bigwedge_{j=1}^{k}$ Büchi(T_j). Its dual objective is the *disjunctive co-Büchi objective* given by $\bigvee_{j=1}^{k}$ coBüchi(T_j).

A *generalized reactivity-1* (GR(1)) objective is an implication between two generalized Büchi objectives; that is, when the generalized Büchi objectives are given by $\bigwedge_{\ell=1}^{k_1}$ Büchi(L_ℓ) and $\bigwedge_{j=1}^{k_2}$ Büchi(U_j), then the GR(1) objective is satisfied if either $\bigvee_{\ell=1}^{k_1}$ coBüchi(L_ℓ) holds, or $\bigwedge_{j=1}^{k_2}$ Büchi(U_j) holds.

Parity objectives. Parity objectives with $c \leq n$ priorities (also called "colors") are defined with respect to a *priority function* $\alpha : V \to \{0, \ldots, c-1\}$ that assigns an integer priority between 0 and $c - 1$ to each vertex. Technically, we define two parity objectives that can easily be transformed into each other by adding one to each priority: A play is in the *even-parity* objective when the *highest* priority occurring infinitely often in the play is *even* and, respectively, a play is in the *odd-parity* objective when the *highest* priority occurring infinitely often is *odd* (equivalently the lowest priority could be used). In parity games the two players are typically called player $\mathcal{E}$ (for even) and player $\mathcal{O}$ (for odd), where the objective of player $\mathcal{E}$ is an even-parity objective and the objective of player $\mathcal{O}$ is an odd-parity objective. From now on we call both types simply *parity objectives* and note that the dual of a parity objective is again a parity objective. We denote by P_i the set of vertices with priority i, i.e., $P_i = \{v \mid \alpha(v) = i\}$. Note that if P_i is empty for some $0 < i < c - 1$, then the priorities $> i$ can be decreased by 2 without changing the parity condition, and when P_{c-1} is empty, we simply have a parity game with one priority less; thus we assume for technical convenience $P_i \neq \emptyset$ for $0 < i < c$. Parity objectives are special cases of both Rabin and Streett objectives [see e.g. Cha07] and of mean-payoff objectives [see e.g. Kla02].

Mean-payoff objectives. For mean-payoff objectives we consider graphs where each edge (u, v) is assigned a weight $w(u, v)$. For the analysis of reactive systems it is sufficient to allow only non-negative integer weights. The mean-payoff of a play $\langle v_0, v_1, v_2, \ldots \rangle$ is defined as $\limsup_{\ell \to \infty} \frac{1}{\ell} \sum_{i=1}^{\ell} w(v_i, v_{i+1})$ and a play is in the objective if it achieves the smallest possible mean-payoff. In this thesis we only consider mean-payoff objectives on graphs, where the algorithmic question reduces to finding the cycle with the smallest average edge weight.

2.4 Winning Sets

A play is *winning* for player z if it is in her objective ϕ. In *graphs* all vertices are owned by player 1 and a vertex v belongs to the *winning set* if there exists an infinite path starting at v that is in the objective. The algorithmic problem in graphs is to determine the winning set, and possibly to construct a corresponding strategy for a given vertex in the winning set.

In *MDPs*, for any measurable set of plays $A \subseteq \Omega$, let $\Pr_v^\sigma(A)$ denote the probability that a play starting at $v \in V$ belongs to A when player 1 plays strategy σ. A strategy σ is *almost-sure (a.s.) winning* from a vertex $v \in V$ for an objective ϕ if $\Pr_v^\sigma(\phi) = 1$. The *almost-sure winning set* $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \phi)$ of player 1 is the set of vertices for which player 1 has an almost-sure winning strategy. The algorithmic problem in the *qualitative analysis* of MDPs is to determine the almost-sure winning set, and possibly a corresponding winning strategy for player 1.

In *game graphs* a strategy σ is winning for player z at a start vertex v if the resulting play is winning for player z irrespective of the strategy of her opponent, player $\overline{z}$, i.e., $\omega(v, \sigma, \pi) \in \phi$ for all $\pi \in \Pi$. A vertex v belongs to the *winning set* W_z of player z if player z has a winning strategy from v. Every vertex is winning for exactly one of the two players and thus the winning sets W_1 and W_2 form a partition of V [Mar75]. When an explicit reference to a specific game $(\mathcal{G}, \phi)$ or game graph $\mathcal{G}$ is required, we use $W_z(\mathcal{G}, \phi)$ and $W_z(\mathcal{G})$, respectively, to refer to the winning sets. The algorithmic problem in game graphs is to determine the winning sets of the two players, and possibly corresponding winning strategies.

2.5 Reachable and Closed Subgraphs

In this section we introduce the basic algorithmic concepts to determine for each of the models whether plays can reach or stay in certain parts of the model.

2.5.1 Graphs: Reachability and SCCs

Reachability. Let GraphReach(G, S) be the set of vertices of G that *can reach* a vertex of $S \subseteq V$ within G. The set GraphReach(G, S) can be found by a linear-time graph exploration.

Strongly connected components (SCCs). For a set of vertices $S \subseteq V$ we denote by $G[S] = (S, E \cap (S \times S))$ the subgraph of the graph G induced by the vertices of S. An induced subgraph $G[S]$ is strongly connected if there exists a path in $G[S]$ between every pair of vertices of S. Also a subgraph induced by a single vertex without a self-loop is considered strongly connected. A *strongly connected component* (*SCC*) of G is a set of vertices $C \subseteq V$ such that the induced subgraph $G[C]$ is strongly connected and C is a maximal set in V with this property. We call an SCC *trivial* if it only contains a single vertex and no edges; and *non-trivial* otherwise. The SCCs of G partition its vertices and can be found in linear time [Tar72]. A bottom SCC C in a directed graph G is an SCC with no edges from vertices of C to vertices of $V \setminus C$, i.e., an SCC without *outgoing* edges. When the algorithm of Tarjan [Tar72] is initialized with a vertex of a bottom SCC, then it has the property to output the bottom SCC in time linear in the number of edges in the bottom SCC. A top SCC is a bottom SCC of *RevG*, i.e., an SCC without *incoming* edges (corresponding to a source in the DAG of SCCs). For more intuition for bottom and top SCCs, consider the graph in which each SCC is contracted into a single vertex (ignoring edges within an SCC). In the resulting directed acyclic graph the sinks represent the bottom SCCs and the sources the top SCCs. Note that every graph has at least one bottom and at least one top SCC. If the graph is not strongly connected, then there exist at least one top and at least one bottom SCC that are disjoint and thus one of them contains at most half of the vertices of G.

2.5.2 MDPs: Random Attractors and MECs

Random attractors. In an MDP $P = ((V, E), (V_1, V_R), \delta)$ the *random attractor* $Attr_R(P, U)$ of a set of vertices $U \subseteq V$ is defined as $Attr_R(P, U) = \bigcup_{i \geq 0} Z_i$ where $Z_0 = U$ and for all $i \geq 0$ let

$$Z_{i+1} = Z_i \cup \{v \in V_R \mid Out(v) \cap Z_i \neq \emptyset\} \cup \{v \in V_1 \mid Out(v) \subseteq Z_i\}. \tag{2.1}$$

Intuitively, the random attractor of U is the set of vertices from which there is a positive probability to reach U; or in other words, the set of vertices from which player 1 cannot almost-surely avoid to reach U. The random attractor $A = Attr_R(P, U)$ can be computed in $O(\sum_{v \in A} Indeg(v))$ time [Bee80; Imm81].

Maximal end-components (MECs). Let $X \subseteq V$ be a vertex set without outgoing random edges, i.e., with $Out(v) \subseteq X$ for all $v \in X \cap V_R$. A sub-MDP of an MDP P induced by a vertex set $X \subseteq V$ without outgoing random edges is defined as $P[X] = ((X, E \cap (X \times X)), (V_1 \cap X, V_R \cap X), \delta)$. Note that the requirement that X has no outgoing random edges is necessary in order to use the same probabilistic transition function δ. An *end-component* (EC) of an MDP P is a set of vertices $X \subseteq V$ such that (a) has no outgoing random edges, i.e., $P[X]$ is a valid sub-MDP, (b) the induced sub-MDP $P[X]$ is strongly connected, and (c) $P[X]$ contains at least one edge. Intuitively, an end-component is a set of vertices for which player 1 can ensure that the play stays within the set and almost-surely reaches all vertices in the set (infinitely often). An end-component is a *maximal end-component* (MEC) if it is maximal under set inclusion. An end-component is *trivial* if it consists of a single vertex (with a self-loop), otherwise it is *non-trivial.* The *MEC-decomposition* of an MDP consists of all MECs of the MDP and the set of vertices that do not belong to any MEC.

2.5.3 Game Graphs: Attractors and Closed Sets

Attractors. In a game graph $\mathcal{G}$ a *z-attractor* $Attr_z(\mathcal{G}, U)$ of a set $U \subseteq V$ is the set of vertices from which player z has a strategy to reach U against all strategies of player $\overline{z}$. A z-attractor can be constructed inductively as follows: Let $Z_0 = U$; and for all $i \geq 0$ let

$$Z_{i+1} = Z_i \cup \{v \in V_z \mid Out(v) \cap Z_i \neq \emptyset\} \cup \{v \in V_{\overline{z}} \mid Out(v) \subseteq Z_i\}.$$

Then $Attr_z(\mathcal{G}, U) = \bigcup_{i \geq 0} Z_i$. The *z-rank* of a vertex v w.r.t. a set U is given by $rank_z(\mathcal{G}, U, v) = \min\{i \mid v \in Z_i\}$ if $v \in Attr_z(\mathcal{G}, U)$ and is ∞ otherwise. Let $A = Attr_z(\mathcal{G}, U)$. The set A can be computed in $O(\sum_{v \in A} Indeg(v))$ time [Bee80; Imm81] and from each vertex of A player z has a memoryless strategy that stays within A to reach U against any strategy of player $\overline{z}$ [Zie98].

Closed sets. A set $U \subseteq V$ is *z-closed* in $\mathcal{G}$ if for all player-z vertices u in U we have $Out(u) \subseteq U$ and for all player-$\overline{z}$ vertices v in U there exists a vertex $w \in Out(v) \cap U$. For each z-closed set U player $\overline{z}$ has a strategy from each vertex of U to keep the play within U, namely choosing an edge (v, w) with $w \in Out(v) \cap U$ whenever the current vertex v is in $U \cap V_{\overline{z}}$ [Zie98]. For a game graph $\mathcal{G}$ and a z-closed set U we denote by $\mathcal{G}[U]$ the game graph induced by the set of vertices U. Note that given that in $\mathcal{G}$ each vertex has at least one outgoing edge, the same property holds for $\mathcal{G}[U]$. We further use the shortcut $\mathcal{G} \setminus X$ to denote $\mathcal{G}[V \setminus X]$ for a set of vertices X such that $V \setminus X$ is z-closed.

The following lemma summarizes two well-known facts about attractors and closed sets that we use frequently.

Lemma 2.5.1. (1) *Let $\mathcal{G}$ be a game graph in which each vertex has at least one outgoing edge and let $U \subseteq V$. Then the set $V \setminus Attr_z(\mathcal{G}, U)$ is z-closed in $\mathcal{G}$ [Zie98, Lemma 4].*

(2) *Let U be z-closed in $\mathcal{G}$. Then $Attr_{\overline{z}}(\mathcal{G}, U)$ is z-closed [Zie98, Lemma 5].*

2.6 Winning Subgraphs

In all our algorithms that determine winning sets for prefix-independent objectives, we first identify certain winning subgraphs and then obtain the winning set by additional attractor or reachability computations. The intuition behind this approach is that an infinite path ω satisfies the prefix-independent objective by visiting some vertices of the underlying graph, denoted by $\mathrm{Inf}(\omega)$, infinitely often. Thus we can first identify sets of vertices $S \subseteq V$ such that the winning player has a strategy to stay within $G[S]$, can visit all vertices of S infinitely often, and for which we have that every ω with $\mathrm{Inf}(\omega) = S$ is in the objective; and then determine the vertices from which the winning player can ensure to reach such sets. In the remainder of this section we define such winning subgraphs for graphs, MDPs, and game graphs and show the correctness of this approach.

2.6.1 Graphs: Good Components

We only provide an intuitive description for graphs as all definitions and properties follow from the more general case of MDPs. Let ϕ be some prefix-independent objective. A set of vertices $X \subseteq V$ is a *good ϕ component* if it 1) induces a non-trivial strongly connected subgraph and 2) each infinite path ω with $\mathrm{Inf}(\omega) = X$

satisfies the objective ϕ. Since ϕ is prefix-independent, every vertex that can reach a good ϕ component is winning. Also the other direction holds: let $\omega^* \in \Omega$ be a play in the objective. Since the graph is finite, after some finite prefix of ω^* only vertices of $\text{Inf}(\omega^*)$ occur in ω^*. As every sequence of vertices in ω^* corresponds to a sequence of edges in the graph, the set $\text{Inf}(\omega^*)$ induces a non-trivial strongly connected subgraph. Thus by $\omega^* \in \phi$ the set $\text{Inf}(\omega^*)$ is a good ϕ component. Hence we can compute the winning set for graphs with a prefix-independent objective ϕ by first determining all (maximal) good ϕ components and then returning the set of vertices that can reach a vertex in a good ϕ component. Since reachability in graphs is in linear time, the running time is dominated by the detection of good ϕ components. A strategy to satisfy the objective starting from some vertex in the winning set can be constructed from the path to reach a good ϕ component and a path that traverses every vertex of the good ϕ component at least once.

2.6.2 MDPs: Good End-Components

In MDPs with prefix-independent objectives we determine the almost-sure (a.s.) winning set by first computing all maximal *good end-components* and then the a.s. winning set for the reachability objective with the union of the good end-components as target set. In this section we show the correctness of this approach (see also [BK08, Chap. 10.6.3]). We define a good end-component as an end-component for which the objective is satisfied if exactly the vertices of the end-component are visited infinitely often.

Definition 2.6.1 (Good end-component)**.** *Given an MDP P and an objective ϕ, an end-component X of P such that each path $\omega \in \Omega$ with* $\text{Inf}(\omega) = X$ *is in ϕ is called a* good ϕ end-component.

The importance of end-components lies in the following property: Once a play reaches an end-component, player 1 can keep the play within the end-component forever. Furthermore, she can visit each vertex in the end-component almost surely and also almost surely infinitely often (Lemma 2.6.2). This implies that in a good end-component player 1 has an a.s. winning strategy (Lemma 2.6.3) and thus player 1 has an a.s. winning strategy from every vertex that can almost-surely reach a good end-component (Lemma 2.6.4 and Corollary 2.6.5). This shows the soundness of the approach of determining the a.s. winning set for a prefix-independent objective by computing the a.s. winning set for a reachability objective with the union of all good end-components as target set.

Lemma 2.6.2. *Given an MDP P and an end-component X, player 1 has a strategy from each vertex of X to reach all vertices of X almost-surely infinitely often while visiting only vertices of X.*

Proof. We define a strategy σ as follows: Choose some arbitrary numbering of the vertices of X. The strategy (with memory) of player 1 is to first follow a shortest path within the end-component (with, say, lexicographic tie breaking) to the first vertex from the current position of the play until this vertex is reached, then a shortest path within the end-component to the second vertex and so on, until she starts with the first vertex again. This is possible because an end-component is a strongly connected subgraph. Since an end-component has no outgoing random edges, the play does not leave the end-component when player 1 plays this strategy. Let $\ell = |X|$ and let α be the smallest positive transition probability in the MDP. Then the probability that the first chosen shortest path is followed with the above strategy is at least α^{ℓ} and the probability that a sequence of ℓ shortest paths within X are followed and thus all vertices of X are visited is at least α^{ℓ^2}. Thus the probability that not all vertices of X have been visited after $q \cdot \ell^2$ steps is at most $(1-\alpha^{\ell^2})^q$, which goes to 0 when q goes to infinity. Hence player 1 has a strategy such that all vertices of X are visited with probability 1. By the same argument, with probability 1 all vertices of X are visited infinitely often because the probability that some vertex is not visited after some finite prefix of length $t \cdot \ell^2$ can be bounded by $(1-\alpha^{\ell^2})^{(q-t)}$. □

Lemma 2.6.3. *From each vertex of a good ϕ end-component X player 1 has a strategy to satisfy ϕ almost-surely.*

Proof. By Lemma 2.6.2 player 1 has a strategy that almost-surely visits all vertices in X but no other vertices infinitely often. By the definition of a good ϕ end-component, all paths visiting exactly the vertices in X infinitely often are in ϕ. Hence, the strategy given by Lemma 2.6.2 is also a.s. winning for ϕ. □

Lemma 2.6.4. *Given an MDP P, a prefix-independent objective ϕ, and a set S of a.s. winning vertices, we have that if $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, Reach(S))$, then also $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \phi)$.*

Proof. Assume $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(S))$ and consider the following strategy. Start with the strategy for reaching S and as soon as one vertex s of S is reached, switch to the a.s. winning strategy of s. As S is reached almost-surely, the vertices visited by the strategy for reaching S do not affect the objective ϕ. □

Corollary 2.6.5 (Soundness of good end-components)**.** *For a prefix-independent objective ϕ and a set of good end-components $\mathcal{X}$ the set $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \mathcal{X}} X)\right)$ is contained in $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \phi)$.*

Another conclusion we can draw from the above lemmata is that if a MEC contains a good end-component, then player 1 has an a.s. winning strategy for the whole MEC because she can reach the good end-component almost-surely from every vertex of the MEC. We exploit this observation in the improved algorithm for coBüchi objectives in Section 6.5.4.

Corollary 2.6.6 (of Lemmata 2.6.2 and 2.6.4)**.** *We are given an MDP P and a prefix-independent objective ϕ. If a MEC X contains an a.s. winning vertex (e.g., a good end-component X), then all vertices in X are a.s. winning for player 1.*

To show the completeness of the approach of computing good end-components, we have to argue that every vertex from which player 1 can satisfy the objective almost-surely also has a strategy to reach a good end-component almost-surely. For this we need two rather technical lemmata. The intuition behind Lemma 2.6.7 is that if a random vertex occurs infinitely often on a path, then almost-surely also each of its successors appears infinitely often on that path. Thus we can argue that vertex sets that are reached infinitely often with positive probability are closed under random edges and hence SCCs within such sets of vertices are end-components (Lemma 2.6.8). To show completeness (Proposition 2.6.9) we then use a set of paths in the objective that are reached with positive probability to show that the vertices that these paths use infinitely often form good end-components. A similar proof is given for Büchi objectives in [CY95].

Lemma 2.6.7. *Let Ω_σ for an MDP P and a strategy σ of player 1 be the set of infinite paths starting at some vertex v that are compatible with the strategy σ. Let $a \in V_R$ be a vertex that is reached infinitely often with probability p from v when player 1 follows σ, i.e., a vertex for which we have $\Pr_\sigma(S_a) = p$ for $S_a = \{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega)\}$. Then we have for each successor b of a that the probability of the subset of Ω_σ containing both a and b infinitely often is equal to the probability of the set S_a, i.e., $\Pr_\sigma(S_{ab}) = p$ for $S_{ab} = \{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega)\}$ and $\Pr_\sigma(S_a \setminus S_{ab}) = 0$ with $S_a \setminus S_{ab} = \{\omega \in \Omega_\sigma \mid a \in \mathrm{Inf}(\omega), b \notin \mathrm{Inf}(\omega)\}$.*

Proof. Whenever a play resulting from the strategy σ visits vertex a, then with some constant probability q the play continues in b. Thus the probability that b is visited less than ℓ times while a is visited n times is upper bounded by $(1-q^\ell)^{n/\ell}$ which goes to 0 with increasing n. Thus, we have $\Pr_\sigma(S_a \setminus S_{ab}) = 0$ and hence the probability for the complement set S_{ab} w.r.t. S_a is $\Pr_\sigma(S_{ab}) = p$. □

Lemma 2.6.8. *We are given an MDP P, a strategy σ of player 1, and the set Ω_σ of infinite paths starting at some vertex v that are compatible with the strategy σ. Let $\Omega' \subseteq \Omega_\sigma$ and let S be the set of vertices that is reached infinitely often with positive probability by paths in Ω' when player 1 follows σ, i.e., $S = \{u \mid \Pr_\sigma(\{\omega \mid u \in \mathrm{Inf}(\omega), \omega \in \Omega'\}) > 0\}$. We have that for each non-trivial SCC C of $P[S]$ and each vertex $a \in C \cap V_R$ all successors b of a are contained in C, i.e., C is an end-component of P.*

Proof. Consider an SCC C, a vertex $a \in C \cap V_R$, and a successor b of a. Then by the definition of S and C, $\Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), \omega \in \Omega'\}) = p$ for some $p > 0$ and by Lemma 2.6.7 we get $\Pr_\sigma(\{\omega \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega), \omega \in \Omega'\}) = p$, i.e., $b \in S$. Note that for each $\omega \in \Omega$ the set $\mathrm{Inf}(\omega)$ induces a strongly connected subgraph. Thus for each of the paths ω in the set $\{\omega \mid a \in \mathrm{Inf}(\omega), b \in \mathrm{Inf}(\omega), \omega \in \Omega'\}$ we have a path from b to a consisting solely of vertices in $\mathrm{Inf}(\omega)$. Since there are just finitely many paths from b to a in P, at least one must have non-zero probability and thus is also contained in S. Hence, b belongs to the SCC C. □

Proposition 2.6.9 (Completeness of good end-components)**.** *Let P be an MDP, let ϕ be a prefix independent objective, and let $\mathcal{X}$ be the set of all good ϕ end-components. Then $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \phi)$ is contained in $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\cup_{X \in \mathcal{X}} X)\right)$.*

Proof. For a vertex $v \in \langle\!\langle 1 \rangle\!\rangle_{as}(P, \phi)$, fix a strategy σ of player 1 such that the objective is satisfied almost-surely. Let P_σ be the sub-MDP of P that consists of the vertices that are visited infinitely often with non-zero probability when player 1 follows strategy σ, starting from v. Note that by Lemma 2.6.8 each SCC of P_σ is an end-component of P. Moreover, σ is a strategy for almost-surely reaching P_σ (each infinite path has to visit at least one vertex infinitely often).

The SCCs of P_σ are not necessarily good end-components but might consist of multiple good end-components, and player 1 might choose one of these good end-components depending on the history of the play. We complete the proof by showing that each vertex of P_σ is contained in a good end-component.

To this end, let Ω_σ be the set of infinite paths starting at v that are compatible with the strategy σ and satisfy the objective. For each set $S \subseteq V$, let Ω_σ^S denote the maximal subset of Ω_σ such that for all $\omega \in \Omega_\sigma^S$ we have $\mathrm{Inf}(\omega) = S$. For an arbitrary vertex u of P_σ we consider all sets Ω_σ^S with $u \in S$. At least one of these sets Ω_σ^S has non-zero probability since there are only finitely many possible sets S and u is visited infinitely often with non-zero probability. Let us consider one of the sets of paths Ω_σ^S with non-zero probability. By Lemma 2.6.8 with $\Omega' = \Omega_\sigma^S$, the set S is closed under random edges. Moreover, as in each path $\omega \in \Omega_\sigma$ the vertices $\mathrm{Inf}(\omega)$ induce a strongly connected sub-graph, the sub-MDP $P[S]$ is also strongly connected and thus S is an end-component. Finally, as the paths $\omega \in \Omega_\sigma^S$ satisfy the objective and the objective ϕ is determined by $\mathrm{Inf}(\omega) = S$, the set S forms a good end component. Hence, we have shown that each vertex of P_σ is contained in a good ϕ end-component, which completes the proof. □

Searching for good end-components. Several of our algorithms maintain vertex sets that are candidates for good end-components. For such a vertex set S we (a) refine the maintained sets according to the SCC decomposition of $P[S]$ and (b) for a set of vertices Y for which we know that it cannot be contained in a good end-component, we remove its random attractor from S. The following lemma shows the correctness of these operations.

Lemma 2.6.10. *Given an MDP $P = (G = (V, E), (V_1, V_R), \delta)$, let X be an end-component with $X \subseteq S$ for some $S \subseteq V$.*

(*a*) *For one SCC C of $G[S]$ we have $X \subseteq C$ and*

(*b*) *for each $Y \subseteq V \setminus X$ and each sub-MDP P' containing X we have $X \subseteq S \setminus Attr_R(P', Y) = \emptyset$.*

Proof. Property (a) holds since every end-component induces a strongly connected sub-MDP. We prove Property (b) by showing that $Attr_R(P', Y)$ does not contain a vertex of X by induction over the recursive definition of a random attractor. Let the sets Z_i be as in Equation (2.1). We have $Z_0 = Y$ and thus $Z_0 \cap X = \emptyset$. Assume we have $Z_i \cap X = \emptyset$ for some $i \geq 0$. No vertex of $V_R \cap X$ has an outgoing edge to $V \setminus X$ and thus the set $X \cap \{v \in V_R \mid Out(P', v) \cap Z_i \neq \emptyset\}$ is empty. Further every vertex in $V_1 \cap X$ has an outgoing edge to a vertex in X. Hence also $X \cap \{v \in V_1 \mid Out(P', v) \subseteq Z_i\}$ is empty and we have that $Z_{i+1} \cap X = \emptyset$. $\square$

2.6.3 Game Graphs: Dominions

In game graphs we search for winning subgraphs called *dominions*. The notion of dominions was introduced by [JPZ08]. A set of vertices $D \neq \emptyset$ is a *player-z dominion* if player z has a winning strategy from every vertex of D that also ensures only vertices of D are visited. Note that a player-z dominion is also a $\overline{z}$-closed set and that the z-attractor of a player-z dominion is again a player-z dominion. The following lemma summarizes some well-known facts about dominions and winning sets.

Lemma 2.6.11. *The following assertions hold for game graphs $\mathcal{G}$ with at least one outgoing edge per vertex and prefix independent objectives. Let $U \subseteq V$.*

(1) *Let U be z-closed in $\mathcal{G}$. Then a $\overline{z}$-dominion in $\mathcal{G}[U]$ is a $\overline{z}$-dominion in $\mathcal{G}$ [JPZ08, Lemma 4.4][1].*

(2) *The set $W_z(\mathcal{G})$ is a z-dominion [JPZ08, Lemma 4.1].*

(3) *Let U be a subset of the winning set $W_z(\mathcal{G})$ of player z and let A be its z-attractor $Attr_z(\mathcal{G}, U)$. Then the winning set $W_z(\mathcal{G})$ of the player z is the union of A and the winning set $W_z(\mathcal{G}[V \setminus A])$, and the winning set $W_{\overline{z}}(\mathcal{G})$ of the opponent $\overline{z}$ is equal to $W_{\overline{z}}(\mathcal{G}[V \setminus A])$ [JPZ08, Lemma 4.5].*

Proof. (1) Player z cannot leave the set z-closed set U, thus player $\overline{z}$ can use the same winning strategy for the vertices of the $\overline{z}$-dominion in $\mathcal{G}$ as in $\mathcal{G}[U]$.

(2) Recall that the winning sets of the two players partition the vertices [Mar75]. Thus we have that for any prefix independent objective that as soon as the play leaves the winning set of player z, the opponent $\overline{z}$ can play his winning strategy starting from the vertex in his winning set that was reached. Hence the winning strategy of player z for the vertices in $W_z(\mathcal{G})$ has to ensure that only vertices of $W_z(\mathcal{G})$ are visited and thus the set $W_z(\mathcal{G})$ is a z-dominion.

(3) By Lemma 2.5.1 (1) the set $V \setminus A$ is z-closed. By (2) $W_{\overline{z}}(\mathcal{G}[V \setminus A])$ is a $\overline{z}$-dominion in $\mathcal{G}[V \setminus A]$ and by (1) also in $\mathcal{G}$. Thus we have $W_{\overline{z}}(\mathcal{G}[V \setminus A]) \subseteq W_{\overline{z}}(\mathcal{G})$. Since $W_z(\mathcal{G})$ and $W_{\overline{z}}(\mathcal{G})$ form a partition of V, we complete the proof by showing $W_z(\mathcal{G}[V \setminus A]) \subseteq W_z(\mathcal{G})$. For this we construct a winning strategy for player z from the vertices of $W_z(\mathcal{G}[V \setminus A])$ in $\mathcal{G}$. As long as the play stays

[1] Note that a $\overline{z}$-closed set of [JPZ08] corresponds to our notion of a z-closed set.

within $V \setminus A$, player z follows her winning strategy in $\mathcal{G}[V \setminus A]$. If the play reaches $A \setminus U$, she follows her attractor strategy to U. When the play reaches U, she follows her winning strategy for U in $\mathcal{G}$ (that exists by assumption). We have that player z wins by the winning strategy in $\mathcal{G}[V \setminus A]$ if the play forever stays within $V \setminus A$ and by the winning strategy for U in $\mathcal{G}$ if the play ever reaches a vertex of A. □

2.7 Conditional Lower Bounds

While classical complexity results are based on assumptions about relationships between complexity classes, e.g., $P \neq NP$, polynomial lower bounds are often based on widely believed, conjectured lower bounds for well studied algorithmic problems. A *conditional lower bound* for a problem B that is based on a conjectured lower bound for a problem A is shown by a *fine-grained reduction* from problem A to problem B.

Fine-grained reductions. Fine-grained reductions relate running time improvements for one algorithmic problem to another (ignoring lower-order terms). The notion of a fine-grained reduction was introduced for cubic running times by [VW10] and formally generalized to arbitrary running times by [Car+16]. For the sake of this thesis the following definition is sufficient: Let (P, T) denote a problem P together with a time complexity T. (P_1, T_1) is *fine-grained reducible* ($\leq_{FGR}$) to (P_2, T_2), if an algorithm with running time $O(T_2^{1-\varepsilon})$ for any $\varepsilon > 0$ for problem P_2 implies an algorithm with running time $O(T_1^{1-\delta})$ for some $\delta > 0$ for problem P_1. Assume we have $(P_1, T_1) \leq_{FGR} (P_2, T_2)$. If it is conjectured that no $O(T_1^{1-\delta})$ for any $\delta > 0$ exists for problem P_1, then we have that there is no $O(T_2^{1-\varepsilon})$ for any $\varepsilon > 0$ for problem P_2 if the conjecture is true. In this case we say that there is a *conditional lower bound* of $\Omega(T_2^{1-o(1)})$ for problem P_2 under the conjecture that there is no $O(T_1^{1-\delta})$ algorithm for P_1 and any $\delta > 0$.

Conjectured lower bounds. We next discuss the popular conjectures that are the basis for the conditional lower bounds in this thesis, see Chapter 1 for pointers to other popular conjectures. The polynomial-time conjectures assume the Word RAM model with $O(\log n)$ bit words.

Remark 2.7.1. *The conjectures that no* polynomial *improvements over the best known running times are possible do not exclude improvements by sub-polynomial factors such as poly-logarithmic factors or factors of, e.g., $2^{\sqrt{\log n}}$ as in [Wil14a]. Similarly, conjectures about exponential running times do not exclude improvements by polynomial factors.*

First, we consider conjectures on Boolean matrix multiplication [VW10; AV14] and triangle detection [AV14] in graphs, which build the basis for our lower bounds on dense graphs. A triangle in a graph is a triple x, y, z of distinct vertices such that $(x, y), (y, z), (z, x) \in E$.

Boolean matrices can be multiplied with any integer matrix multiplication algorithm; for this $O(n^{\omega})$ time algorithms with $\omega < 2.3727$ are known [Vas12; Le 14]. However, due to the high constants hidden in the running time bound, these "algebraic" algorithms are currently impractical. Therefore, other types, i.e., "combinatorial", algorithms are desirable. In the conjectures below, combinatorial means avoiding the existing types of sub-cubic time matrix multiplication algorithms. There exist combinatorial algorithms that improve upon the basic cubic time algorithm by logarithmic factors, e.g., the $O(n^3/\log^4 n)$ time algorithm for Boolean matrix multiplication and triangle detection by Yu [Yu15]. With fast matrix multiplication, triangles can be detected in time $O(\min(m^{2\omega/(\omega+1)}, n^{\omega}))$ [AYZ97].

Conjecture 2.7.2 (Combinatorial Boolean matrix multiplication conjecture (BMM)). *There is no $O(n^{3-\varepsilon})$ time combinatorial algorithm for computing the Boolean product of two $n \times n$ Boolean matrices for any $\varepsilon > 0$.*

Conjecture 2.7.3 (Combinatorial triangle conjecture (CTC)). *There is no $O(n^{3-\varepsilon})$ time combinatorial algorithm that can detect whether a graph contains a triangle for any $\varepsilon > 0$.*

By a result of Vassilevska Williams and Williams [VW10], we have that BMM is equivalent to CTC. A weaker assumption, without the restriction to combinatorial algorithms, is that detecting a triangle in a graph takes super-linear time. A generalization of the triangle conjecture is the k-clique conjecture [Woe04; ABV15a].

Second, we consider the Strong Exponential Time Hypothesis [IPZ01; CIP09] and the orthogonal vectors conjecture [AVW16], the former dealing with the satisfiability of a propositional logic formula in conjunctive normal form (CNF-SAT) (see [Abb+16] for conjectures on other representations of the satisfiability problem), and the latter with the *orthogonal vectors problem.*

The orthogonal vectors problem (OV). Given two sets S_1, S_2 of d-bit vectors with $|S_1|, |S_2| \leq N$ and $d \in \Theta(\log N)$, are there $u \in S_1$ and $v \in S_2$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$?

For the orthogonal vectors problem the current fastest algorithm runs in time $O(n^{2-1/O(\log(d/\log n))})$ [AWY15]. For k-CNF-SAT, where the number of literals per clause is bounded by a constant k, there exist several algorithms with a running time of the form $O(2^{n(1-c/k)})$ for some constant c [DH09]. For the general CNF-SAT problem an improvement over exhaustive search is a major open problem.

Conjecture 2.7.4 (Strong Exponential Time Hypothesis (SETH)). *For each $\varepsilon > 0$ there is a k such that k-CNF-SAT on n variables and m clauses cannot be solved in time $O(2^{(1-\varepsilon)n} \operatorname{poly}(m))$.*

Conjecture 2.7.5 (Orthogonal vectors conjecture (OVC)). *There is no $O(N^{2-\varepsilon})$ time algorithm for the orthogonal vectors problem for any $\varepsilon > 0$.*

By a result of Williams [Wil05] we know that SETH implies OVC, i.e., whenever a problem is hard assuming OVC, it is also hard when assuming SETH. Hence, it is preferable to use OVC for proving lower bounds. Finally, to the best of our knowledge, no such relations between the former two conjectures and the latter two conjectures are known.

CHAPTER 3

Approximating the Minimum Cycle Mean

3.1 Introduction

In this chapter we show the first approximation algorithm for mean-payoff objectives on graphs that improves the dependence of the running time on the number of vertices n, compared to the best known exact algorithm. For mean-payoff objectives a good component (see Section 2.6.1) is a strongly connected subgraph (induced by a vertex set), i.e., a not necessarily simple cycle, with minimum average edge weight among all cycles. Therefore computing the winning set for mean-payoff objectives on graphs reduces to computing the *minimum cycle mean* for each SCC of the input graph and then determining the set of vertices that can reach the SCCs where the minimum cycle mean is the smallest. A strategy to satisfy the mean-payoff objective is then given by a corresponding cycle with minimum mean weight and a path from a start vertex to the cycle.

Minimum cycle mean problem. The input to the problem is a finite directed graph $G = (V, E, w)$ with a set V of n vertices, a set E of m edges, and a weight function w that assigns an integer weight to every edge. Given a cycle C, the mean weight $\mu(C)$ of the cycle is the ratio of the sum of the weights of the cycle and the number of edges in the cycle. The algorithmic question asks to compute $\mu = \min\{\mu(C) \mid C \text{ is a cycle}\}$: the minimum cycle mean. The minimum cycle mean problem is an important problem in combinatorial optimization and has a long history of algorithmic study. An $O(nm)$ time algorithm for the problem was given by Karp [Kar78]. The current best known algorithm for the problem by Orlin and Ahuja, which is over two decades old, requires $O(m\sqrt{n}\log(nW))$ time [OA92], where W is the maximum absolute value of the weights.

Applications. The minimum cycle mean problem is a basic combinatorial optimization problem that has numerous applications in network flows [AMO93]. In the context of the formal analysis of systems, the performance of systems as well as the average resource consumption of systems is modeled as the minimum cycle mean problem. For quantitative objectives such as mean-payoff a system is modeled as a *weighted* directed graph, where vertices represent states of the system, edges represent transitions, and every edge is assigned a *non-negative* integer representing the resource consumption (or delay) associated with the transition. The computation of a minimum average resource consumption behavior (or minimum average response time) corresponds to the computation of the minimum cycle mean. Several recent works model other quantitative aspects of system analysis (such as robustness) also as the mean-weight problem (also known as *mean-payoff* problem) [Blo+09; DM10].

Results. This chapter contains the following results.

(1) *Reduction to min-plus matrix multiplication.* We show that the minimum cycle mean problem is reducible to the problem of a logarithmic number of min-plus matrix multiplications of $n \times n$-matrices, where n is the number of vertices of the graph. Our result implies that algorithmic improvements for min-plus matrix multiplication will carry over to the minimum cycle mean problem with a logarithmic multiplicative factor in the running time.

(2) *Faster approximation algorithm.* When the weights are non-negative, we present the first $(1+\epsilon)$-approximation algorithm for the problem that outputs $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\epsilon)\mu$. Our algorithm runs in time $\widetilde{O}(n^{\omega} \log^3 (nW/\epsilon)/\epsilon)$, where $O(n^{\omega})$ is the time required for the *classic* $n \times n$-matrix multiplication. The current best known bound for ω is $\omega < 2.3727$ [Vas12; Le 14]. As usual, the $\widetilde{O}$-notation is used to "hide" a polylogarithmic factor, i.e., $\widetilde{O}(T(n,m,W)) = O(T(n,m,W) \cdot \text{polylog}(n))$.

For the computation of $\hat{\mu}$, $O(n^2)$ space is needed. If $O(n^2 \log(nW/\epsilon))$ space is used instead, i.e., the intermediate results of the approximation algorithm are saved, we can additionally output a cycle with mean weight at most $\hat{\mu}$.

The worst-case complexity of the current best known algorithm for the minimum cycle mean problem is $O(m\sqrt{n} \log (nW))$ [OA92], which could be as bad as $O(n^{2.5} \log (nW))$. Thus for a $(1+\epsilon)$-approximation our algorithm provides better dependence on n.

Note that in applications related to the analysis of system the weights are always non-negative (they represent resource consumption, delays, etc); and the weights are typically small, whereas the state space of the system is large. Moreover, due to imprecision in modeling, approximations in weights are already introduced during the modeling phase. Hence a $(1+\epsilon)$-approximation of the minimum cycle mean problem with small weights and large graphs is a relevant algorithmic problem in the formal analysis of systems, and we improve the long-standing complexity of the problem.

Table 3.1: Current fastest asymptotic running times for the minimum cycle mean

Reference	Running time	Approx.	Weight Range
Karp [Kar78]	$O(mn)$	exact	$[-W, W]$
Orlin and Ahuja [OA92]	$O(m\sqrt{n}\log(nW))$	exact	$[-W, W] \cap \mathbb{Z}$
Sankowski [San05] (implicit)	$\widetilde{O}(Wn^{\omega}\log(nW))$	exact	$[-W, W] \cap \mathbb{Z}$
Butkovic and Cuninghame-Green [BC92]	$O(n^2)$	exact	$\{0, 1\}$
Theorem 3.4.7	$\boldsymbol{\widetilde{O}(n^{\omega}\log^3(nW/\epsilon)/\epsilon)}$	$\boldsymbol{1+\epsilon}$	$[\mathbf{0}, \boldsymbol{W}] \cap \mathbb{Z}$

The key technique that we use to obtain the approximation algorithm is a combination of the value iteration algorithm for the minimum cycle mean problem, and a technique used for an approximation algorithm for all-pair shortest path problem for directed graphs. Table 3.1 compares our algorithm with the asymptotically fastest existing algorithms.

Outline. In the rest of this section we discuss related work and motivate the minimum cycle mean problem by its relation to negative cycle detection. We summarize all needed definitions in Section 3.2. In Section 3.3 we describe how min-plus matrix multiplication can be used to compute the minimum cycle mean exactly. In Section 3.4 we present our approximation algorithm and prove its correctness and running time. In Section 3.5 we show how at the cost of storing the intermediate results an approximately optimal cycle can be output.

3.1.1 Related work

We distinguish two types of algorithms: algorithms that are independent of the weights of the graph and algorithms that depend on the weights in some way. By W we denote the maximum absolute edge weight of the graph. Recall that graphs with mean-payoff objectives are special cases of both MDPs with mean-payoff objectives and mean-payoff games [EM79; GKK88; ZP96; Ras16].

Algorithms independent of weights. The classic algorithm of Karp [Kar78] uses a dynamic programming approach to find the minimum cycle mean and runs in time $O(mn)$. A corresponding cycle can easily be computed given the outcome of the algorithm. The main drawback of Karp's algorithm is that its best-case and worst-case running times are the same. The algorithms of Hartmann and Orlin [HO93] and of Dasdan and Gupta [DG98] address this issue, but also have a worst-case complexity of $O(mn)$. By solving the more general parametric shortest path problem, Karp and Orlin [KO81] can compute the minimum cycle mean in time $O(mn \log n)$. Young, Tarjan, and Orlin [YTO91] improve this running time to $O(mn + n^2 \log n)$.

A well known algorithm for MDPs with mean-payoff objectives is the value iteration algorithm [FV97]. In each iteration this algorithm spends time $O(m)$ and in total it performs $O(nW)$ iterations. Madani [Mad02] showed that on graphs a certain variant of the value iteration algorithm "converges" to the optimal cycle after $O(n^2)$ iterations, which gives a running time of $O(mn^2)$ for computing the minimum cycle mean. Using similar ideas he also obtains a running time of $O(mn)$. Howard's policy iteration algorithm is another well-known algorithm for MDPs with mean-payoff objectives [How60]. The complexity of this algorithm for graphs is unresolved. Recently, Hansen and Zwick [HZ10] provided a class of weighted graphs on which Howard's algorithm performs $\Omega(n^2)$ iterations where each iteration takes time $O(m)$. For a summary of recent results on Howard's algorithm see [Mil13].

Algorithms depending on weights. If a graph is complete and has only two different edge weights, then the minimum cycle mean problem can be solved in time $O(n^2)$ because the matrix of its weights is bivalent [BC92].

Another approach is to use the connection to the problem of detecting a negative cycle. Lawler [Law76] gave a reduction for finding the minimum cycle mean that performs $O(\log(nW))$ calls to a negative cycle detection algorithm. The main idea is to perform binary search on the minimum cycle mean. In each search step the negative cycle detection algorithm is run on a graph with modified edge weights. Orlin and Ahuja [OA92] extend this idea by the approximate binary search technique [Zem87]. By combining approximate binary search with their scaling algorithm for the assignment problem, they can compute the minimum mean cycle in time $O(m\sqrt{n} \log nW)$.

Note that in its full generality the single-source shortest paths problem (SSSP) also demands the detection of a negative cycle reachable from the source vertex.[1] Therefore it is also possible to reduce the minimum cycle mean problem to SSSP. The best time bounds on SSSP are as follows. Goldberg's scaling algorithm [Gol95] solves the SSSP problem (and therefore also the negative cycle detection problem) in time $O(m\sqrt{n} \log W)$. McCormick [McC93] combines approximate binary search with Goldberg's scaling algorithm to an $O(m\sqrt{n} \log nW)$ time algorithm for the minimum cycle mean problem, which matches the result of Orlin and Ahuja [OA92]. Sankowski's matrix multiplication based algorithm [San05] solves the SSSP problem in time $\widetilde{O}(Wn^\omega)$. By combining binary search with Sankowski's algorithm, the minimum cycle mean problem can be solved in time $\widetilde{O}(Wn^\omega \log nW)$.

Approximation of minimum cycle mean. To the best of our knowledge, our algorithm is the first approximation algorithm specifically for the minimum cycle mean problem. There are both additive and multiplicative fully polynomial-time

[1]Remember that, for example, Dijkstra's algorithm for computing single-source shortest paths requires non-negative edge weights which excludes the possibility of negative cycles.

approximation schemes for solving mean-payoff games [Rot+10; Bor+11], which is a more general problem. Note that in contrast to finding the minimum cycle mean it is not known whether the exact solution to a mean-payoff game can be computed in polynomial time. The results of [Rot+10] and [Bor+11] are obtained by reductions to a pseudo-polynomial algorithm for solving mean-payoff games. In the case of the minimum cycle mean problem, these reductions do not provide an improvement over the current fastest exact algorithms mentioned above.

Min-plus matrix multiplication. Our approach reduces the problem of finding the minimum cycle mean to computing the (approximate) min-plus product of matrices. The naive algorithm for computing the min-plus product of two matrices runs in time $O(n^3)$. To date, no algorithm is known that runs in time $O(n^{3-\alpha})$ for some $\alpha > 0$, i.e., (truly) subcubic time. This is in contrast to classic matrix multiplication that can be done in time $O(n^\omega)$, where the current best bound on ω is $\omega < 2.3727$ [Vas12; Le 14]. Moreover, Vassilevska Williams and Williams [VW10] showed that computing the min-plus product is computationally "equivalent" to a series of problems including all-pairs shortest paths and negative triangle detection in the following sense: if one of these problems has a subcubic algorithm, then all of them have. This provides evidence for the hardness of these problems.

Still, the running time of $O(n^3)$ for the min-plus product can be improved by sub-polynomial factors. Fredman [Fre76] gave an algorithm for computing the min-plus product with a running time of $O(n^3 (\log \log n)^{1/3} / (\log n)^{1/3})$. After a long line of improvements, Chan [Cha10] presented an algorithm of similar flavor with a running time of $O(n^3 (\log \log n)^3 / (\log n)^2)$. Recently, Williams [Wil14b] developed a randomized algorithm that runs in time $O(n^3 / 2^{\Omega(\log n / \log \log n)^{1/2}})$ and is correct with high probability. Williams also gave a deterministic version that runs in time $O(n^3 / 2^{\log^\delta n})$ for some $\delta > 0$.

A different approach for computing the min-plus product of two integer matrices is to reduce the problem to classic matrix multiplication [Yuv76]. In this way, the min-plus product can be computed in the pseudo-polynomial time of $O(M n^\omega \log M)$ [AGM97]. This observation was used by Alon, Galil, and Margalit [AGM97] and Zwick [Zwi02] to obtain faster all-pairs shortest paths algorithms in directed graphs for the case of small integer edge weights. Zwick also combines this min-plus matrix multiplication algorithm with an adaptive scaling technique that allows to compute $(1 + \epsilon)$-approximate all-pairs shortest paths in graphs with non-negative edge weights. Our approach of finding the minimum cycle mean extensively uses this technique.

3.1.2 Relation to negative cycle detection

In the following we provide additional motivation for our approach of *approximating* the minimum cycle mean by relating it to negative cycle detection. A solution to the minimum cycle mean problem immediately gives a solution to the negative cycle detection problem. Therefore, an improved running time for finding the minimum cycle mean will also give an improved running time for detecting a negative cycle, which in turn has numerous applications [WT05] and comes up as a subproblem in algorithms for other problems, such as the minimum-cost flow problem [CG99]. However, researchers are stuck with finding faster algorithms for negative cycle detection. Even more, by the binary-search based reduction of minimum cycle mean to negative cycle detection, the worst-case running times of both problems are the same, up to a factor of $O(\log nW)$. Approximation helps to break the running time barrier induced by negative cycle detection. Intuitively, the approximation provided by our algorithm is not good enough to distinguish between a positive and a negative cycle. Therefore our approximation algorithm can be faster than known algorithms for the negative cycle detection problem.

Our new algorithm needs two non-standard assumptions. First, it only works for graphs with non-negative edge weights. Second, it provides a multiplicative approximation. Both of these assumptions are necessary for bypassing the negative cycle detection problem—only one of them is not enough. On the one hand, if we could compute the minimum cycle mean exactly for non-negative edge weights, then, by shifting of weights, we could solve the negative cycle detection problem. On the other hand, a reduction of Gentilini [Gen14] (initially designed for mean-payoff games) shows that if one can compute a multiplicative ϵ-approximation $\hat{\mu}$ of the minimum cycle mean μ such that $|(\hat{\mu} - \mu)/\mu| \leq \epsilon$, then one can immediately detect negative cycles. In particular, this is true for $\epsilon \leq 1$ for which μ and $\hat{\mu}$ will always have the same sign. To see this, note that when $\hat{\mu}$ and μ have different signs, then $|(\hat{\mu} - \mu)/\mu| = (|\hat{\mu}| + |\mu|)/|\mu| = |\hat{\mu}|/|\mu| + 1 > 1$. Thus, the only hope of getting a multiplicative ϵ-approximation for arbitrary edge weights without solving the negative cycle detection problem is when $\epsilon > 1$. We remark that Gentilini's definition of an ϵ-approximation is a generalization of our definition of a $(1 + \epsilon)$-approximation to arbitrary edge weights.

3.2 Definitions

Throughout this chapter we let $G = (V, E, w)$ be a finite, weighted directed graph with a set of vertices V and a set of edges E such that every vertex has at least one outgoing edge. The weight function w assigns a non-negative integer weight to every edge. We denote by n the number of vertices of G and by m the number of edges of G. Note that $m \geq n$ because every vertex has at least one outgoing edge.

A *path* is a finite sequence of edges $P = (e_1, \dots, e_t)$ such that for all consecutive edges $e_i = (x_i, y_i)$ and $e_{i+1} = (x_{i+1}, y_{i+1})$ of P we have $y_i = x_{i+1}$. Note that edges may be repeated on a path, we *do not* only consider simple paths. The *length $|P|$ of a path $P = (e_1, \dots, e_t)$* is the number of edges of P, i.e. $|P| = t$. The *weight of a path $P = (e_1, \dots, e_t)$*, denoted by $w(P)$, is the sum of its edge weights, i.e. $w(P) = \sum_{1 \leq i \leq t} w(e_i)$. The *mean weight* of the path P is the ratio $w(P)/|P|$. A *cycle* is a path in which the start vertex and the end vertex are the same. In a *simple cycle* each vertex contained in the cycle appears in exactly two of the edges of the cycle;

thus the length of a simple cycle can be at most n.

The *minimum cycle mean* of G is the minimum mean weight of any cycle in G. For every vertex x we denote by $\mu(x)$ the value of the minimum mean-weight cycle reachable from x. The minimum cycle mean of G is simply the minimum $\mu(x)$ over all vertices x.

We will use that there always exists a *simple* cycle with minimum mean weight and thus we can assume that the cycle with minimum mean weight has at most n edges. This is already used implicitly in [Kar78]. We show first that every non-simple cycle can be partitioned into a set of simple cycles, which already appeared in [ZP96].

Proposition 3.2.1 ([ZP96]). *Let P be a path in the graph $G = (V, E)$ from x to y. Let $G_P = (V, E_P)$ be the multigraph consisting of the edges in P. Then E_P can be partitioned into a simple path from x to y and a set S of simple cycles.*

Proof. Initialize the set S with the empty set. Follow the path P until a vertex is encountered for the second time. Let v be this vertex and let C be the set of edges between the first and the second encounter of v. Then C is a simple cycle since no other vertex was encountered twice. Add C to S, remove C from P and follow the updated path P', starting again from vertex x, until a vertex is encountered for the second time. Repeat this removal of simple cycles until the final vertex of P is reached without encountering any vertex twice. Then the remaining path is a simple path from x to y. □

Proposition 3.2.2. *Given a set of simple cycles S with total mean weight*

$$\mu = \frac{\sum_{C_i \in S} \sum_{e \in C_i} w(e)}{\sum_{C_i \in S} |C_i|},$$

there exists a simple cycle in S with mean weight at most μ.

Proof. Denote for each simple cycle $C_i \in S$ its mean weight by μ_i and its number of edges by m_i. Then

$$\mu \sum_{C_i \in S} m_i = \frac{\sum_{C_i \in S} \sum_{e \in C_i} w(e)}{\sum_{C_i \in S} m_i} \sum_{C_i \in S} m_i = \sum_{C_i \in S} w(C_i) = \sum_{C_i \in S} \mu_i m_i \geq \min_{C_i \in S}(\mu_i) \cdot \sum_{C_i \in S} m_i$$

and thus $\min_{C_i \in S}(\mu_i) \leq \mu$. □

Corollary 3.2.3. *Let μ be the minimum cycle mean of a graph G. Then there exists a* simple *cycle in G with mean weight μ.*

For every vertex x and every integer $t \geq 1$ we denote by $\delta_t(x)$ the minimum weight of all paths starting at x that have length t, i.e., consist of exactly t edges. For all pairs of vertices x and y and every integer $t \geq 1$ we denote by $d_t(x, y)$ the minimum weight of all paths of length t from x to y. If no such path exists we set $d_t(x, y) = \infty$.

For every matrix A we denote by $A[i, j]$ the entry at the i-th row and the j-th column of A. We only consider $n \times n$ matrices with integer entries, where n is the size of the graph. We assume that the vertices of G are numbered consecutively from 1 to n, which allows us to use $A[x, y]$ to refer to the entry of A belonging to vertices x and y. The *weight matrix D of G* is the matrix containing the weights of G. For all pairs of vertices x and y we set $D[x, y] = w(x, y)$ if the graph contains the edge (x, y) and $D[x, y] = \infty$ otherwise.

We denote the *min-plus product* of two matrices A and B by $A \star B$. The min-plus product is defined as follows. If $C = A \star B$, then for all indices $1 \leq i, j \leq n$ we have $C[i, j] = \min_{1 \leq k \leq n}(A[i, k] + B[k, j])$. We denote by A^t the t-th power of the matrix A. Formally, we set $A^1 = A$ and $A^{t+1} = A \star A^t$ for $t \geq 1$. We denote by ω the exponent of classic matrix multiplication, i.e., the product of two $n \times n$ matrices can be computed in time $O(n^\omega)$. The current best bound on ω is $\omega < 2.3727$ [Vas12; Le 14].

3.3 Reduction to Min-Plus Matrix Multiplication

In the following we explain the main idea of our approach, which is to use min-plus matrix multiplication to find the minimum cycle mean. The well-known value iteration algorithm uses a dynamic programming approach to compute in each iteration a value for every vertex x from the values of the previous iteration. After t iterations, the value computed by the value iteration algorithm for vertex x is equal to $\delta_t(x)$, the minimum weight of all paths with length t starting at x. We are actually interested in $\mu(x)$, the value of the minimum mean-weight cycle reachable from x. It is well known that $\lim_{t \to \infty} \delta_t(x)/t = \mu(x)$ and that the value of $\mu(x)$ can be computed from $\delta_t(x)$ if t is large enough; specifically, for $t = 4n^3 W$ the unique number in $\left(\delta_t(x)/t - 1/[2n(n-1)], \delta_t(x)/t + 1/[2n(n-1)]\right) \cap \mathbb{Q}$ that has a denominator of at most n is equal to $\mu(x)$ [ZP96]. Thus, one possibility to determine $\mu(x)$ is the following: first, compute $\delta_t(x)$ for t large enough with the value iteration algorithm and then compute $\mu(x)$ from $\delta_t(x)$. However, using the value iteration algorithm for computing $\delta_t(x)$ is expensive because its running time is linear in t and thus pseudo-polynomial.

Our idea is to compute $\delta_t(x)$ for a large value of t by using fast matrix multiplication instead of the value iteration algorithm. We will compute the matrix D^t, the t-th power of the weight matrix (using min-plus matrix multiplication). The matrix D^t contains the value of the minimum-weight path of length exactly t for all pairs of vertices. Given D^t, we can determine the value $\delta_t(x)$ for every vertex x by finding the minimum entry in the row of D^t corresponding to x.

Proposition 3.3.1. *For every $t \geq 1$ and all vertices x and y we have (i) $d_t(x, y) = D^t[x, y]$ and (ii) $\delta_t(x) = \min_{y \in V} D^t[x, y]$.*

Proof. We give the proof for the sake of completeness of the presentation. The claim $d_t(x, y) = \boldsymbol{D}^t[x, y]$ follows from a simple induction on t. If $t = 1$, then clearly the minimal-weight path of length 1 from x to y is the edge from x to y if it exists, otherwise $d_t(x, y) = \infty$. If $t \geq 1$, then a minimal-weight path of length t from x to y (if it exists) consists of some outgoing edge of $e = (x, z)$ as its first edge and then a minimal-weight path of length $t - 1$ from z to y. We therefore have $d_t(x, y) = \min_{(x,z)\in E} w(x, z) + d_{t-1}(z, y)$. By the definition of the weight matrix and the induction hypothesis we get $d_t(x, y) = \min_{z\in V} \boldsymbol{D}[x, z] + \boldsymbol{D}^{t-1}[z, y]$. Therefore the matrix $\boldsymbol{D} \star \boldsymbol{D}^{t-1} = \boldsymbol{D}^t$ contains the value of $d_t(x, y)$ for every pair of vertices x and y.

For the second claim, $\delta_t(x) = \min_{y\in V} \boldsymbol{D}^t[x, y]$, observe that by the definition of $\delta_t(x)$ we obviously have $\delta_t(x) = \min_{y\in V} d_t(x, y)$ because the minimal-weight path of length t starting at x has *some* node y as its end point. □

Using this approach, the main question is how fast the matrix $\boldsymbol{D}^t$ can be computed. The most important observation is that $\boldsymbol{D}^t$ (and therefore also $\delta_t(x)$) can be computed by repeated squaring with only $O(\log t)$ min-plus matrix multiplications. This is different from the value iteration algorithm, where t iterations are necessary to compute $\delta_t(x)$.

Proposition 3.3.2. *For every $t \geq 1$ we have $\boldsymbol{D}^{2t} = \boldsymbol{D}^t \star \boldsymbol{D}^t$. Therefore the matrix $\boldsymbol{D}^t$ can be computed with $O(\log t)$ many min-plus matrix multiplications.*

Proof. We give the proof for the sake of completeness of the presentation. It can easily be verified that the min-plus matrix product is associative [AHU74] and therefore $\boldsymbol{D}^{2t} = \boldsymbol{D}^t \star \boldsymbol{D}^t$. Therefore, if t is a power of two, we can compute $\boldsymbol{D}^t$ with $\log_2 t$ min-plus matrix multiplications. If t is not a power of two, we can decompose $\boldsymbol{D}^t$ into $\boldsymbol{D}^t = \boldsymbol{D}^{t_1} \star \cdots \star \boldsymbol{D}^{t_k}$ where each $t_i \leq t$ (for $1 \leq i \leq k$) is a power of two and $k \leq \lceil \log_2 t \rceil$. By storing intermediate results, we can compute $\boldsymbol{D}^{2^i}$ for every $0 \leq i \leq \lceil \log_2 t \rceil$ with $\lceil \log_2 t \rceil$ min-plus matrix multiplications. Using the decomposition above, we have to multiply at most $\lceil \log_2 t \rceil$ such matrices to obtain $\boldsymbol{D}^t$. Therefore the total number of min-plus matrix multiplications needed for computing $\boldsymbol{D}^t$ is $O(\log t)$. □

The running time of this algorithm depends on the time needed for computing the min-plus product of two integer matrices. This running time usually depends on the two parameters n and M, where n is the size of the $n \times n$ matrices to be multiplied (in our case this is equal to the number of vertices of the graph) and the parameter M denotes the maximum absolute integer entry in the matrices to be multiplied. When we multiply the matrix $\boldsymbol{D}$ by itself to obtain $\boldsymbol{D}^2$, we have $M = W$, where W is the maximum absolute edge weight. However, M increases with every multiplication and in general we can bound the maximum absolute integer entry of the matrix $\boldsymbol{D}^t$ only by $M = tW$. Note that $O(n^2)$ operations are necessary to extract the minimum cycle mean $\mu(x)$ for all vertices x from the matrix $\boldsymbol{D}^t$ (with $t = O(n^3 W)$ [ZP96], see above).

Theorem 3.3.3. *If the min-plus product of two $n \times n$ matrices with entries in $[-M, M] \cap \mathbb{Z} \cup \{\infty\}$ can be computed in time $T(n, M)$, then the minimum cycle mean problem can be solved in time $T(n, tW) \log_2 t$ where $t = O(n^3 W)$.*

Note that necessarily $T(n, M) = \Omega(n^2)$ because the result matrix has n^2 entries that have to be written.

Unfortunately, the approach outlined above does not immediately improve the running time for the minimum cycle mean problem because min-plus matrix multiplication currently cannot be done fast enough. However, our approach is still useful for solving the minimum cycle mean problem *approximately* because approximate min-plus matrix multiplication can be done faster than its exact counterpart.

3.4 Approximation Algorithm

In this section we design an algorithm that computes an approximation of the minimum cycle mean in graphs with nonnegative integer edge weights. It follows the approach of reducing the minimum cycle mean problem to min-plus matrix multiplication outlined in Section 3.3. The key to our algorithm is a fast procedure for computing the min-plus product of two integer matrices approximately. We will proceed as follows. First, we explain how to compute an approximation F of D^t, the t-th power of the weight matrix D. From this we get, for every vertex x, an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$, the minimum-weight of all paths of length t starting at x. We then argue that for t large enough (in particular $t = O(n^2 W/\epsilon)$), the value $\delta_t(x)/t$ is an approximation of $\mu(x)$, the minimum cycle mean of cycles reachable from x. By combining both approximations we can show that $\hat{\delta}_t(x)/t$ is an approximation of $\mu(x)$. Thus, the main idea of our algorithm is to compute an approximation of D^t for a large enough t.

3.4.1 Computing an Approximation of D^t

Our first goal is to compute an approximation of the matrix D^t, the t-th power of the weight matrix D, given $t \geq 1$. Zwick [Zwi02] provides the following algorithm for approximate min-plus matrix multiplication.

Theorem 3.4.1 ([Zwi02]). *Let A and B be two $n \times n$ matrices with integer entries in $[0, M]$ and let $C := A \star B$. Let $R \geq \log_2 n$ be a power of two. The algorithm* approx-min-plus(A, B, M, R) *computes the approximate min-plus product $\overline{C}$ of A and B in time*[2] $O(n^\omega R \log(M) \log^2(R) \log(n))$ *such that for every $1 \leq i, j \leq n$ it holds that $C[i, j] \leq \overline{C}[i, j] \leq (1 + 4/R) C[i, j]$.*

[2]The running time of approx-min-plus is given by $O(n^\omega \log M)$ times the time needed to multiply two $O(R \log n)$-bit integers. With the Schönhage-Strassen algorithm for large integer multiplication, two k-bit integers can be multiplied in $O(k \log k \log\log k)$ time, which gives a running time of $O(n^\omega R \log(M) \log(n) \log(R \log n) \log\log(R \log n))$. This can be bounded by the running time given in Theorem 3.4.1 if $R \geq \log n$, which will always be the case in the following.

We now give a modification (see Algorithm 3.1) of Zwick's algorithm for approximate shortest paths [Zwi02] such that, given an ϵ in $(0, 1]$, the algorithm computes a $(1+\epsilon)$-approximation F of D^t when t is a power of two such that for all $1 \leq i, j \leq n$ we have $D^t[i,j] \leq F[i,j] \leq (1+\epsilon)D^t[i,j]$. Just as we can compute D^t exactly with $\log_2 t$ min-plus matrix multiplications, the algorithm computes the $(1+\epsilon)$-approximation of D^t in $\log_2 t$ iterations. However, in each iteration only an approximate min-plus product is computed. Let F_s be the approximation of $D_s := D^{2^s}$. In the s-th iteration we use approx-min-plus$(F_{s-1}, F_{s-1}, tW, R)$ to calculate F_s with R chosen beforehand such that the desired error bound is reached for $F = F_{\log_2 t}$.

Algorithm 3.1: Approximation of D^t

```
input  : weight matrix $D$, error bound $\epsilon \in (0,1]$, $t$ (a power of 2)
output : $(1+\epsilon)$-approximation of $D^t$
1 $F \leftarrow D$
2 $r \leftarrow 4\log_2 t / \ln(1+\epsilon)$
3 $R \leftarrow 2^{\lceil \log_2 r \rceil}$
4 for $\log_2 t$ times do
5     $F \leftarrow$ approx-min-plus$(F, F, 2tW, R)$
6 return $F$
```

Lemma 3.4.2. *Given an $0 < \epsilon \leq 1$ and a power of two $t \geq 1$, Algorithm 3.1 computes a $(1+\epsilon)$-approximation F of D^t in time*

$$O\left(n^\omega \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW)\log^2\left(\frac{\log(t)}{\epsilon}\right)\log(n)\right) = \widetilde{O}\left(n^\omega \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW)\right)$$

such that $D^t[i,j] \leq F[i,j] \leq (1+\epsilon)\, D^t[i,j]$ for all $1 \leq i, j \leq n$.

Proof. We start with the main idea of the proof and continue with the details afterwards. The running time of approx-min-plus depends linearly on R and logarithmically on M, the maximum entry of the input matrices. Algorithm 3.1 calls approx-min-plus $\log_2 t$ times. Each call increases the error by a factor of $(1 + 4/R)$. However, as only $\log_2 t$ approximate matrix multiplications are used, setting R to the smallest power of 2 that is larger than $4\log_2(t)/\ln(1+\epsilon)$ will suffice to bound the approximation error by $(1+\epsilon)$. We will show that $2tW$ is an upper bound on the entries in the input matrices for approx-min-plus. The stated running time will follow from these two facts and Theorem 3.4.1.

Let F_s be the approximation of $D_s := D^{2^s}$ computed by the algorithm after iteration s. Recall that 2^sW is an upper bound on the maximum entry in D_s. As we will show, all entries in F_s are at most $(1+\epsilon)$ times the entries in D_s. Since we assume $\epsilon \leq 1$, we have $1+\epsilon \leq 2$. Thus $2^{s+1}W$ is an upper bound on the entries in F_s. Hence for F_s with $1 \leq s < \log_2 t$, i.e., for all input matrices of approx-min-plus in our algorithm, $2tW$ is an upper bound its entries.

This results in an overall running time of

$$\begin{aligned}
&O\left(n^{\omega} R \log(tW) \log^2(R) \log(n) \cdot \log(t)\right) ,\\
&= O\left(n^{\omega} \cdot \frac{\log^2(t)}{\log(1+\epsilon)} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\log(1+\epsilon)}\right) \log(n)\right) ,\\
&= O\left(n^{\omega} \cdot \frac{\log^2(t)}{\epsilon} \cdot \log(tW) \log^2\left(\frac{\log(t)}{\epsilon}\right) \log(n)\right) .
\end{aligned}$$

The last equation follows from the inequality $\ln(x) \leq x - 1$ for $x > 0$: With $x = 1/(1+\epsilon)$ and $\epsilon > 0$ we have $1/\ln(1+\epsilon) \leq (1+\epsilon)/\epsilon$. Since $\epsilon \leq 1$ it follows that $1/\log(1+\epsilon) = O(1/\epsilon)$.

To show the claimed approximation guarantee, we will prove that the inequality

$$D_s[i,j] \leq F_s[i,j] \leq \left(1 + \frac{4}{R}\right)^s D_s[i,j] \tag{3.1}$$

holds after the s-th iteration of Algorithm 3.1 by induction on s. Note that the $(1+\epsilon)$-approximation follows from this inequality because the parameter R is chosen such that after the $(\log_2 t)$-th iteration of the algorithm it holds that

$$\left(1 + \frac{4}{R}\right)^{\log_2 t} \leq \left(1 + \frac{\ln(1+\epsilon)}{\log_2 t}\right)^{\log_2 t} \leq e^{\ln(1+\epsilon)} = 1 + \epsilon . \tag{3.2}$$

For $s = 0$ we have $F_s = D_s$ and the inequality holds trivially. Assume the inequality holds for s. We will show that it also holds for $s + 1$.

First we prove the lower bound on $F_{s+1}[i,j]$. Let C_{s+1} be the exact min-plus product of F_s with itself, i.e., $C_{s+1} = F_s \star F_s$. Let k_c be the minimizing index such that $C_{s+1}[i,j] = \min_{1 \leq k \leq n}(F_s[i,k] + F_s[k,j]) = F_s[i,k_c] + F_s[k_c,j]$. By the definition of the min-plus product

$$D_{s+1}[i,j] = \min_{1 \leq k \leq n} (D_s[i,k] + D_s[k,j]) \leq D_s[i,k_c] + D_s[k_c,j] . \tag{3.3}$$

By the induction hypothesis and the definition of k_c we have

$$D_s[i,k_c] + D_s[k_c,j] \leq F_s[i,k_c] + F_s[k_c,j] = C_{s+1}[i,j] . \tag{3.4}$$

By Theorem 3.4.1 the values of F_{s+1} can only be larger than the values in C_{s+1}, i.e.,

$$C_{s+1}[i,j] \leq F_{s+1}[i,j] . \tag{3.5}$$

Combining inequalities (3.3), (3.4), and (3.5) yields the claimed lower bound,

$$D_{s+1}[i,j] \leq F_{s+1}[i,j] .$$

Next we prove the upper bound on $F_{s+1}[i,j]$. Let k_d be the minimizing index such that $D_{s+1}[i,j] = D_s[i,k_d] + D_s[k_d,j]$. Theorem 3.4.1 gives the error from one call of approx-min-plus, i.e., the error in the entries of F_{s+1} compared to the entries of C_{s+1}. We have

$$F_{s+1}[i,j] \le \left(1+\frac{4}{R}\right) C_{s+1}[i,j]\,. \tag{3.6}$$

By the definition of the min-plus product we know that

$$C_{s+1}[i,j] \le F_s[i,k_d] + F_s[k_d,j]\,. \tag{3.7}$$

By the induction hypothesis and the definition of k_d we can reformulate the error obtained in the first s iterations of Algorithm 3.1 as follows:

$$\begin{aligned} F_s[i,k_d] + F_s[k_d,j] &\le \left(1+\frac{4}{R}\right)^s D_s[i,k_d] + \left(1+\frac{4}{R}\right)^s D_s[k_d,j] \\ &= \left(1+\frac{4}{R}\right)^s \left(D_s[i,k_d] + D_s[k_d,j]\right) \\ &= \left(1+\frac{4}{R}\right)^s D_{s+1}[i,j]\,. \end{aligned} \tag{3.8}$$

Combining inequalities (3.6), (3.7), and (3.8) yields the upper bound

$$F_{s+1}[i,j] \le \left(1+\frac{4}{R}\right)^{s+1} D_{s+1}[i,j]\,. \qquad \square$$

Once we have computed an approximation of the matrix D^t, we extract from it the minimal entry of each row to obtain an approximation of $\delta_t(x)$. Here we use the equivalence between the minimum entry of row x of D^t and $\delta_t(x)$ established in Proposition 3.3.1. Remember that $\delta_t(x)/t$ approaches $\mu(x)$ for t large enough and later on we want to use the approximation of $\delta_t(x)$ to obtain an approximation of the minimum cycle mean $\mu(x)$.

Lemma 3.4.3. *The value $\hat{\delta}_t(x) := \min_{y\in V} F[x,y]$ approximates $\delta_t(x)$ with $\delta_t(x) \le \hat{\delta}_t(x) \le (1+\epsilon)\delta_t(x)$.*

Proof. Let y_f and y_d be the indices where the x-th rows of F and D^t obtain their minimal values, respectively, i.e.,

$$y_f := \arg\min_{y\in V} F[x,y] \quad \text{and} \quad y_d := \arg\min_{y\in V} D^t[x,y]\,.$$

By these definitions and Lemma 3.4.2 we have

$$\delta_t(x) = D^t[x,y_d] \le D^t[x,y_f] \le F[x,y_f] = \hat{\delta}_t(x)$$

and

$$\hat{\delta}_t(x) = F[x,y_f] \le F[x,y_d] \le (1+\epsilon) D^t[x,y_d]\,. \qquad \square$$

3.4.2 Approximating the Minimum Cycle Mean

We now add the next building block to our algorithm. So far, we can obtain an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ for any t that is a power of two. We now show that $\delta_t(x)/t$ is itself an approximation of the minimum cycle mean $\mu(x)$ for t large enough. Then we argue that also $\hat{\delta}_t(x)/t$ approximates the minimum cycle mean $\mu(x)$ for t large enough. This value of t bounds the number of iterations of our algorithm. A similar technique was also used in [ZP96] to bound the number of iterations of the value iteration algorithm for mean-payoff games.

We start by showing that $\delta_t(x)/t$ differs from $\mu(x)$ by at most nW/t *for any* t. Then we turn this additive error into a multiplicative error by choosing a large enough value of t. A multiplicative error implies that we have to compute the solution exactly for $\mu(x) = 0$. We will use a separate procedure to identify all vertices x with $\mu(x) = 0$ and compute the approximation only for the remaining vertices. Note that $\mu(x) > 0$ implies $\mu(x) \geq 1/n$ because all edge weights are integers and we can assume by Corollary 3.2.3 that the cycle with minimum mean weight has at most n edges.

Lemma 3.4.4. *For every $x \in V$ and every integer $t \geq 1$ it holds that*

$$t \cdot \mu(x) - nW \leq \delta_t(x) \leq t \cdot \mu(x) + nW\,.$$

Proof. We first show the lower bound on $\delta_t(x)$. Let P be a path of length t starting at x with weight $\delta_t(x)$. Consider the cycles in P and let E' be the multiset of the edges in P that are in a cycle of P. There can be at most n edges that are not in a cycle of P, thus there are at least $\max(t-n, 0)$ edges in E'. Since $\mu(x)$ is the minimum mean weight of any cycle reachable from x, the sum of the weights of the edges in E' can be bounded below by $\mu(x)$ times the number of edges in E'. Furthermore, the value of $\mu(x)$ can be at most W. As we only allow non-negative edge weights, the sum of the weights of the edges in E' is a lower bound on $\delta_t(x)$. Thus we have

$$\delta_t(x) \geq \sum_{e \in E'} w(e) \geq (t-n)\mu(x) \geq t \cdot \mu(x) - n \cdot \mu(x) \geq t \cdot \mu(x) - nW\,.$$

Next we prove the upper bound on $\delta_t(x)$. Let l be the length of the shortest path from x to a vertex y in a minimum mean-weight cycle C reachable from x (such that only y is both in the shortest path and in C). Let c be the length of C. By Corollary 3.2.3 we can assume that C is a simple cycle. Let the path Q be a path of length t that consists of the shortest path from x to y, $\lfloor (t-l)/c \rfloor$ rounds on C, and $t - l - c\lfloor (t-l)/c \rfloor$ additional edges in C. By the definition of $\delta_t(x)$, we have $\delta_t(x) \leq w(Q)$. The sum of the length of the shortest path from x to y and the number of the remaining edges of Q not in a complete round on C can be at most n because in a graph with non-negative weights no shortest path has a cycle and no vertices in C except y are contained in the shortest path from x to y. Each of these edges has a weight of at most W. The mean weight of C is $\mu(x)$, thus the sum of the weights of the edges in all complete rounds on C is $\mu(x) \cdot c\lfloor(t-l)/c\rfloor \le \mu(x) \cdot t$. Hence we have

$$\delta_t(x) \le w(Q) \le t \cdot \mu(x) + nW\,. \qquad \square$$

In the next step we show that we can use the fact that $\delta_t(x)/t$ is an approximation of $\mu(x)$ to obtain a $(1+\epsilon)$-approximation $\hat{\mu}(x)$ of $\mu(x)$ even if we only have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ with $(1+\epsilon)$-error. We exclude the case $\mu(x) = 0$ for the moment.

Lemma 3.4.5. *Assume we have an approximation $\hat{\delta}_t(x)$ of $\delta_t(x)$ such that $\delta_t(x) \le \hat{\delta}_t(x) \le (1+\epsilon)\delta_t(x)$ for $0 < \epsilon \le 1/2$. If*

$$t \ge \frac{n^2 W}{\epsilon}, \quad \mu(x) \ge \frac{1}{n}, \quad \textit{and} \quad \hat{\mu}(x) := \frac{\hat{\delta}_t(x)}{(1-\epsilon)t},$$

then

$$\mu(x) \le \hat{\mu}(x) \le (1+7\epsilon)\mu(x)\,.$$

Proof. We first show that $\hat{\mu}(x)$ is at least as large as $\mu(x)$. From Lemma 3.4.4 we have $\delta_t(x) \ge t \cdot \mu(x) - nW$. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \ge \mu(x) - \frac{nW}{t} \ge \mu(x) - \frac{\epsilon}{n} \ge \mu(x) - \epsilon\mu(x) \ge (1-\epsilon)\mu(x)\,.$$

Thus, by the assumption $\delta_t(x) \le \hat{\delta}_t(x)$ we have

$$\mu(x) \le \frac{\hat{\delta}_t(x)}{(1-\epsilon)t} = \hat{\mu}(x)\,.$$

For the upper bound on $\hat{\mu}(x)$ we use the inequality $\delta_t(x) \le t \cdot \mu(x) + nW$ from Lemma 3.4.4. As t is chosen large enough,

$$\frac{\delta_t(x)}{t} \le \mu(x) + \frac{nW}{t} \le \mu(x) + \frac{\epsilon}{n} \le (1+\epsilon)\mu(x)\,. \tag{3.9}$$

With $\hat{\delta}_t(x) \le (1+\epsilon)\delta_t(x)$ this gives

$$\hat{\mu}(x) = \frac{\hat{\delta}_t(x)}{(1-\epsilon)t} \le \frac{(1+\epsilon)^2}{(1-\epsilon)}\mu(x)\,.$$

It can be verified by simple arithmetic that for $\epsilon > 0$ the inequality $\epsilon \le 1/2$ is equivalent to

$$\frac{(1+\epsilon)^2}{(1-\epsilon)} \le (1+7\epsilon)\,. \qquad \square$$

As a last ingredient to our approximation algorithm, we design a procedure that deals with the special case that the minimum cycle mean is zero. Since our goal is an algorithm with a multiplicative error, we have to be able to compute the solution exactly in this case. This can be done in linear time because the edge-weights are non-negative.

Proposition 3.4.6. *Given a graph with non-negative integer edge weights, we can find all vertices x with $\mu(x) = 0$ and output a cycle with mean weight of zero in time $O(m)$.*

Proof. Note that in the case of non-negative edge weights we have $\mu(x) \geq 0$. Furthermore, a cycle can only have mean weight 0 if all edges on this cycle have weight 0. Thus, it will be sufficient to detect cycles in the graph that only contain edges that have weight 0.

We proceed as follows. Let $G^0 = (V, E^0)$ denote the subgraph of G that only contains edges of weight 0, i.e., $E^0 = \{e \in E | w(e) = 0\}$. As argued above, G contains a zero-mean cycle if and only if G^0 contains a cycle. We can check whether G^0 contains a cycle by computing the strongly connected components of G^0: G^0 contains a cycle if and only if it has a strongly connected component of size at least 2 (we can assume w.l.o.g. that there are no self-loops). Let Z be the set of all vertices in a strongly connected components of G^0 of size at least 2. We can identify all vertices that can reach a zero-mean cycle by performing a linear-time graph traversal to identify all vertices that can reach Z.

To actually output a zero-mean cycle, consider one of the strongly connected components of G^0 of size at least 2 and output a cycle found by a linear-time traversal of the component.

Since all steps take linear time, the total running time of this algorithm is $O(m)$. □

Finally, we wrap up all arguments to obtain our algorithm for approximating the minimum cycle mean. This algorithms performs $\log t$ approximate min-plus matrix multiplications to compute an approximation of D^t and $\delta_t(x)$. Lemma 3.4.5 tells us that $t = n^2W/\epsilon$ is just the right number to guarantee that our approximation of $\delta_t(x)$ can be used to obtain an approximation of $\mu(x)$. The value of t is relatively large but the running time of our algorithm depends on t only in a logarithmic way.

Theorem 3.4.7. *Given a graph with n vertices and non-negative integer edge weights of at most W, we can compute an approximation $\hat{\mu}(x)$ of the minimum cycle mean for every vertex x such that $\mu(x) \leq \hat{\mu}(x) \leq (1+\epsilon)\mu(x)$ for $0 < \epsilon \leq 1$ in time*

$$O\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\epsilon}\right)}{\epsilon}\right)\log(n)\right) = \widetilde{O}\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\right).$$

Proof. First we find all vertices x with $\mu(x) = 0$. By Proposition 3.4.6 this takes time $O(n^2)$ for $m = O(n^2)$. For the remaining vertices x we approximate $\mu(x)$ as follows.

Let $\epsilon' := \epsilon/7$. If we execute Algorithm 3.1 with weight matrix D, error bound ϵ' and t such that t is the smallest power of two with $t \geq n^2W/\epsilon'$, we obtain a $(1+\epsilon')$-approximation $F[x,y]$ of $D^t[x,y]$ for all vertices x and y (Lemma 3.4.2). By calculating for every x the minimum entry of $F[x,y]$ over all y we have a $(1+\epsilon')$-approximation of $\delta_t(x)$ (Lemma 3.4.3). By Lemma 3.4.5 $\hat{\mu}(x) := \hat{\delta}_t(x)/((1-\epsilon')t)$ is for this choice of t an approximation of $\mu(x)$ such that $\mu(x) \leq \hat{\mu}(x) \leq (1+$

$7\epsilon')\mu(x)$. By substituting ϵ' with $\epsilon/7$ we get $\mu(x) \leq \hat{\mu}(x) \leq (1+\epsilon)\mu(x)$, i.e., a $(1+\epsilon)$-approximation of $\mu(x)$.

By Lemma 3.4.2 the running time of Algorithm 3.1 for $t = 2^{\lceil \log(n^2W/\epsilon') \rceil} = O(n^2W/\epsilon)$ is

$$O\left(\frac{n^\omega}{\epsilon}\log^2\left(\frac{n^2W}{\epsilon}\right)\log\left(\frac{n^2W^2}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{n^2W}{\epsilon}\right)}{\epsilon}\right)\log(n)\right).$$

With $\log(n^2W) \leq \log((nW)^2) = O(\log(nW))$ we get that Algorithm 3.1 runs in time

$$O\left(\frac{n^\omega}{\epsilon}\log^3\left(\frac{nW}{\epsilon}\right)\log^2\left(\frac{\log\left(\frac{nW}{\epsilon}\right)}{\epsilon}\right)\log(n)\right). \tag{3.10}$$

□

Note that since the weights are integral and by Corollary 3.2.3 there is a simple cycle with minimum mean weight, the distance between two possible values for the minimum cycle mean is $\geq 1/(n(n-1))$ [ZP96]. Thus the minimum cycle mean could be determined exactly from $\hat{\mu}$ when $\epsilon \leq 1/(n^2W)$. However, this would give a running time that is worse than the known exact algorithms.

3.5 Finding an Approximately Optimal Cycle

In this section we provide an algorithm that outputs, for $\mu > 0$, a cycle with mean weight of at most $(1+\epsilon)\mu$ for ϵ chosen as in Theorem 3.4.7. The algorithm uses the intermediate results of Algorithm 3.1, i.e., the matrices F_s for $1 \leq s \leq \log_2 t$, which approximate D^{2^s}. For $t = O(n^2W/\epsilon)$ it needs $O(n^2\log(nW/\epsilon))$ space compared to the space of $O(n^2)$ for computing only the approximate minimum cycle mean. The running time of the algorithm is $O(n^2\log(nW/\epsilon)/\epsilon)$ plus the running time of Algorithm 3.1, where the former is dominated by the running time of Algorithm 3.1.

The main idea of our algorithm for cycle extraction is as follows. Consider a path P of length t starting at x with approximately minimum weight. Let y be the final vertex of this path. This path can be described as the sum of a simple path from x to y and a set of cycles. If we subtract the simple path, the mean weight of the remaining part might become larger, but not too large because a simple path can have at most n edges, while the whole path has $t = O(n^2W/\epsilon)$ edges. Given a set of cycles and an upper bound on the mean weight of the edges in these cycles, there must be a simple cycle with mean at most this upper bound. If we can efficiently find this cycle, we can output it as a cycle with approximately minimum mean weight. However, spending time proportional to the path length t would already be too costly. Therefore, we partition the path into $O(n/\epsilon)$ non-overlapping *segments* of equal size and consider the corresponding path P' in a graph G' where the only edges are these segments. In G' there are $O(n/\epsilon)$ edges, thus we can find a cycle C in G' with minimum mean weight and at most n edges in $O(n^2/\epsilon)$ time with Karp's algorithm. We can split each edge of G', which corresponds to a segment of the original path P, in half by obtaining the *midpoint* of the segment, i.e., the vertex in the middle of the corresponding path segment. Let G'' be the graph consisting of the segment halves of the edges in the cycle C. Since the cycle C has at most n edges, the graph G'' has at most $2n$ edges. Thus we can run Karp's algorithm on G'' in $O(n^2)$ time. We repeat the halving of the segments until the remaining segments correspond to edges in the original graph and we can indeed output a cycle with approximately minimum mean weight.

In the implementation of the algorithm we need to obtain the midpoint of a path segment. A segment corresponds to an entry $F_s[u, v]$ in the matrix that approximates D^{2^s} for some segment length $l = 2^s$ with $1 \leq s \leq \log_2 t$. Thus we can obtain a midpoint z such that $F_{s-1}[u, z] + F_{s-1}[z, v]$ is approximately equal to $d_l(u, v)$ in time $O(n)$ by computing $\arg\min_{z' \in V}(F_{s-1}[u, z'] + F_{s-1}[z', v])$.

We consider a graph whose edges are exactly the edges of a (non-simple) path. To show that in this graph there exists a simple cycle with mean weight at most the mean weight of all the edges in the path, we view the graph as a multigraph and apply Propositions 3.2.1 and 3.2.2. Note that parts of the path might be traversed multiple times. Thus we need to consider the multiset of edges in the path in order to argue about the ratio of the sum of the weights to the number of edges. The multigraph is only needed in the analysis, not in the actual algorithm.

Lemma 3.5.1. *Given all intermediate results of Algorithm 3.1 for some $0 < \epsilon \leq 1$ and a power of two $t \geq 1$, Algorithm 3.2 outputs a cycle with mean weight $\hat{\mu}$ such that $\hat{\mu} \leq (1 + \epsilon) \min_{x \in V} \hat{\delta}_t(x)/t$ in time $O((n^2/\epsilon) \log t)$.*

Proof. Let F_s denote the intermediate results of Algorithm 3.1 for $1 \leq s \leq \log_2 t$. Further let $(x, y) = \arg\min_{(i,j)}(F_{\log_2 t}[i, j])$, i.e., $x = \arg\min_{x' \in V} \hat{\delta}_t(x')$. Let P be a path from x to y with length t and weight $\delta_t(x) \leq w(P) \leq \hat{\delta}_t(x)$ such that $d_t(x, y) \leq \hat{\delta}_t(x) \leq (1 + \epsilon) d_t(x, y)$. Remember that by Equation (3.1) each entry $F_s[i, j]$ is a $(1 + 4/R)^s$-approximation of $d_{2^s}(i, j)$ for all vertices i, j and $1 \leq s \leq \log_2 t$ and that by Equation (3.2) we have $(1 + 4/R)^{\log_2 t} \leq 1 + \epsilon$. Thus, for any length $l \leq t$ that is a power of 2 and any pair of vertices (u, v) with $d_l(u, v)$, we can obtain a midpoint z such that $d_l(u, v) \leq F_{\log_2(l/2)}[u, z] + F_{\log_2(l/2)}[z, v] \leq (1 + 4/R)^{\log_2 l} d_l(u, v)$ in time $O(n)$ by computing $\arg\min_{z' \in V}(F_{\log_2(l/2)}[u, z'] + F_{\log_2(l/2)}[z', v])$.

In Algorithm 3.2 (lines 5–10) we divide the path P into $O(n/\epsilon)$ segments of equal size l such that l is the largest power of 2 smaller or equal $\epsilon t/(2n)$. We do this by repeatedly doubling the number of segments we split the path P into, starting with the segment (x, y), which represents the whole path P. In each iteration a midpoint for each of the current segments is found, thus in total less than $\sum_{i=0}^{\log_2(4n/\epsilon)} 2^i = O(n/\epsilon)$ midpoints have to be found. Hence splitting path P into $O(n/\epsilon)$ segments takes time $O(n^2/\epsilon)$.

Let E_l be the set of all segments of P and let G_l be the graph with vertices V and edges E_l where the edges have weight $w_l(i, j) = F_{\log_2 l}[i, j]$. We will prove below that there exists a simple cycle in G_l with mean weight at most $\hat{\mu} \leq (1 + \epsilon)\hat{\delta}_t(x)/t$. Given the existence of such a cycle, we can find a cycle C in G_l with at most n

Algorithm 3.2: Cycle with approximately minimum mean weight

```
input : $F_s$ for $s = 1, \dots, \log_2 t$ with $D^{2^s}[i,j] \le F_s[i,j] \le \left(1+\frac{4}{R}\right)^s D^{2^s}[i,j]$ and
        $\left(1+\frac{4}{R}\right)^{\log_2 t} \le (1+\epsilon)$
output: cycle with mean weight $\le (1+\epsilon)\min_{(i,j)} F_{\log_2 t}[i,j]/t$
(x, y) ← arg min_{(i,j)}(F_{log_2 t}[i, j]) and let P denote the corresponding path
$L \leftarrow \epsilon t/(2n)$
$l \leftarrow 2^{\lfloor \log_2 L \rfloor}$
$E' \leftarrow \{(x,y)\}$
for $j \leftarrow 1$ to $t/l$ do                    /* split $P$ into $O(n/\epsilon)$ segments */
    $E'' \leftarrow \emptyset$
    foreach $(u,v) \in E'$ do
        find midpoint $z$ of segment corresponding to $(u,v)$
        $E'' \leftarrow E'' \cup \{(u,z),(z,v)\}$
    $E' \leftarrow E''$
foreach $u, v \in V$ and $s = 1, \dots, \log_2 t$ do
    let $w_{2^s}(u,v) := F_s[u,v]$          /* definition of weight function */
$G' \leftarrow (V, E', w_l)$
find cycle $C$ with minimum mean weight and $\le n$ edges in $G'$
while $l > 1$ do                                   /* cycle refinement */
    $l \leftarrow l/2$
    $E' \leftarrow \emptyset$
    foreach $(u,v) \in C$ do
        find midpoint $z$ of segment corresponding to $(u,v)$
        $E' \leftarrow E' \cup \{(u,z),(z,v)\}$
    $G' \leftarrow (V, E', w_l)$
    find cycle $C$ with minimum mean weight and $\le n$ edges in $G'$
return $C$
```

segments and minimum mean weight $\le \hat{\mu}$ with Karp's algorithm [Kar78] in $O(n^2/\epsilon)$ time (line 14 of Algorithm 3.2) because the number of edges in G_l is $O(n/\epsilon)$.

To show that there exists a simple cycle in G_l with mean weight at most $\hat{\mu} \le (1+\epsilon)\hat{\delta}_t(x)/t$, we consider the multigraph G'_l with vertices V and edges E'_l with weight $w_l(i,j) = F_{\log_2 l}[i,j]$, where E'_l is the multiset of all segments of P. Note that by Theorem 3.4.1 the mean weight of the edges in G'_l is bounded above by $\hat{\delta}_t(x)/t$. Since $|E'_l| \ge n$, we have that G'_l contains at least one cycle. By Proposition 3.2.1, the set E'_l can be partitioned into a simple path from x to y and a set of simple cycles S. The simple path from x to y can contain at most n segments, where each segment represents $l \le \epsilon t/(2n)$ edges in the original graph. Since we assume non-negative edge weights, the mean weight $\hat{\mu}$ of the segments in S is at most

$$\hat{\mu} \le \frac{\hat{\delta}_t(x)}{t - n\frac{\epsilon t}{2n}} \le \frac{1}{1-\frac{\epsilon}{2}}\frac{\hat{\delta}_t(x)}{t} \le (1+\epsilon)\frac{\hat{\delta}_t(x)}{t} .$$

By Proposition 3.2.2 there exists a simple cycle with mean weight at most $\hat{\mu} \leq (1+\epsilon)\hat{\delta}_t(x)/t$ in S and in the multigraph G'_l, and thus in the simple graph G_l.

As explained above, we can find the midpoint z of each segment $(u, v) \in C$ in $O(n)$ time. Thus we can obtain a multigraph $G'_{l/2}$ with vertices V, the two edges (u, z) and (z, v) for each $(u, v) \in C$, and weights $w_{l/2}(i, j) = F_{\log_2(l/2)}[i, j]$ in time $O(n^2)$. By Proposition 3.2.1 the edges in $G'_{l/2}$ can be described as a set of simple cycles. By Proposition 3.2.2 there exists a simple cycle C' in $G'_{l/2}$ with mean weight at most $\hat{\mu}$. Given the existence, we again only have to consider the corresponding simple graph $G_{l/2}$, for which we can obtain a minimum mean cycle with at most n edges with Karp's algorithm in time $O(n^2)$ (line 22 of Algorithm 3.2). We can repeat this process of reducing the segment size for the found simple cycle until each segment corresponds to an edge in the original graph and we thus have indeed found a cycle with mean weight at most $\hat{\mu}$.

Since we halve the segment length in each iteration, at most $\log_2 l$ iterations are needed. Hence the running time of the while-loop (lines 15–22 of Algorithm 3.2) can be bounded by $O(n^2 \log t)$, which implies a total running time of $O((n^2/\epsilon) \log t)$. □

Remark 3.5.2. *In Algorithm 3.2 above we use the naive $O(n)$ time procedure for finding midpoints. A more efficient approach is the following: We modify the procedure* approx-min-plus *in Algorithm 3.1 to additionally output a matrix of* witnesses *as described in [Zwi02]. For $C := F_s \star F_s$ the matrix of witnesses contains for each entry (i, j) in F_{s+1} an index $1 \leq k \leq n$ such that $C[i, j] \leq F_s[i, k] + F_s[k, j] \leq (1 + 4/R)C[i, j]$. Having stored the witness matrices, a midpoint (witness) can be found in constant time.*

Theorem 3.5.3. *Given a graph with n vertices and non-negative integer edge weights of at most W, we can compute a cycle with mean weight at most $\hat{\mu}$ such that $\mu \leq \hat{\mu} \leq (1+\epsilon)\mu$ for $0 < \epsilon \leq 1$ in*

$$\widetilde{O}\left(\frac{n^\omega}{\epsilon} \log^3\left(\frac{nW}{\epsilon}\right)\right) \text{ time and } O\left(n^2 \log\left(\frac{nW}{\epsilon}\right)\right) \text{ space.}$$

Proof. If $\mu = \min_{x' \in V} \mu(x') = 0$, we can find a cycle with mean weight of zero in $O(n^2)$ time by Proposition 3.4.6. If $\mu > 0$, we set $\epsilon' = \epsilon/7$ like in the proof of Theorem 3.4.7. We execute Algorithm 3.1 with weight matrix D, error bound ϵ' and t such that t is the smallest power of two with $t \geq n^2 W/\epsilon'$ and save all intermediate results F_s for $s = 1, \ldots, \log_2 t$. This gives the claimed running time by Theorem 3.4.7 and requires $O(n^2 \log t)$ space.

By Lemma 3.5.1 the running time to output a cycle given the intermediate results of Algorithm 3.1 is subsumed by the running time of Algorithm 3.1. It remains to show the approximation guarantee on the mean weight of the cycle computed by Algorithm 3.2. The lower bound is given trivially since we actually output a cycle in the original graph. Let $x = \arg\min_{x' \in V} \mu(x')$. By Lemma 3.4.4 and the choice of t (Equation (3.9)) we have $\delta_t(x)/t \leq (1+\epsilon')\mu(x)$. Lemma 3.4.3 gives $\hat{\delta}_t(x) \leq$

$(1+\epsilon')\delta_t(x)$. Combined with Lemma 3.5.1 this yields an upper bound on the error of the mean weight $\hat{\mu}$ of the output cycle of

$$\hat{\mu} \leq (1+\epsilon')^3 \mu \leq (1+7\epsilon')\mu \leq (1+\epsilon)\mu \,. \qquad \square$$

3.6 Conclusion

We introduced the problem of *approximating* the minimum cycle mean. It would be interesting to study whether there is a faster approximation algorithm for the minimum cycle mean problem, maybe at the cost of a worse approximation. The running time of our algorithm immediately improves if faster algorithms for classic matrix multiplication, min-plus matrix multiplication or approximate min-plus multiplication are found. However, a different approach might lead to better results and might shed new light on how well the problem can be approximated. Therefore it would be interesting to remove the dependence on fast matrix multiplication and develop a so-called combinatorial algorithm.

CHAPTER 4

Algorithms for MDPs with Streett Objectives

4.1 Introduction

In this chapter we present two algorithms with improved running times for MDPs with Streett objectives, one for "dense" graphs that improves the dependence of the running time on n and one for "sparse" graphs that improves the dependence of the running time on m. Recall from Section 2.3 that a Streett objective for a set of k target pairs $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$ is defined as $\mathrm{Streett(TP)} = \bigwedge_{j=1}^{k}(\mathrm{coBüchi}(L_j) \vee \mathrm{Büchi}(U_j))$, i.e., a play satisfies the objective if for all $1 \leq j \leq k$ either the vertices of L_j are visited only finitely often or some vertex of U_j is visited infinitely often.

Significance. Streett objectives can express all ω-regular languages and are therefore canonical objectives in formal verification. Additionally, Streett objectives arise, for example, in the verification of systems with strong fairness conditions [LH00; DPC09; Fra86]. In program verification, a scheduler is *strongly fair* if every event that is enabled infinitely often is scheduled infinitely often. The verification of closed systems with strong fairness conditions directly corresponds to checking the non-emptiness of Streett automata, which in turn corresponds to determining the winning set in graphs with Streett objectives. With MDPs we can additionally model the probabilistic behavior of systems.

Our results. For an MDP $P = ((V, E), (V_1, V_R), \delta)$ and a set of k target pairs TP the size of the input is measured in terms of $m = |E|$, $n = |V|$, k, and $b = \sum_{j=1}^{k}(|L_j| + |U_j|) \leq 2nk$. Recall that we assume that each vertex has an outgoing and an incoming edge and therefore we have $m \geq n$. In this chapter we prove the following theorem.

Theorem 4.1.1. *For MDPs with Streett objectives the almost-sure winning set can be computed in* $O(\min(n^2, m^{1.5}\sqrt{\log n}) + b \log n)$ *time.*

Relation to existing work. The basic algorithm for MDPs with Streett objectives makes $\min(n, k)$ calls to an algorithm that computes all MECs and therefore, using the MEC algorithms of [CH14], takes time $O(\min(n, k) \cdot (\min(n^2, m^{1.5}) + b))$. The new algorithms improve over the running time of the basic algorithm for all parameter combinations with $k \in \omega(\log n)$. For *graphs* an $O(m \min(\sqrt{m \log n}, k, n) + b \min(\log n, k))$ time algorithm is known [HT96]. We improve over this running time when m is at least of order $n^{4/3} \log^{-1/3} n + b^{2/3} \log^{1/3} n$ and $k \in \omega(n^2/m)$. Further related work for graphs with Streett objectives includes the algorithms of [LH00; DPC09] that only achieve a running time of $O((m + b) \min(n, k))$ but consider additional properties such as "on-the-fly" computation and producing certificates of bounded size.

Technical contribution and outline. *Good end-components.* In Section 2.6 we show that the algorithmic question for MDPs with Streett objectives reduces to identifying *good* (*Streett*) *end-components* (see Definition 2.6.1 for the definition of a good end-component). In this chapter we describe algorithms that identify all (maximal) good end-components.

Basic algorithm (*Section 4.2*). The basic algorithm (folklore) maintains a set of end-components as candidates for good end-components. This set of candidates is initialized with the MECs of the input MDP. The algorithm then repeatedly removes "bad vertices" that cannot be in a good end-component, namely sets L_j for which no vertex of U_j is in the same end-component, and then recomputes the MECs of the remaining MDP.

Improving upon the basic algorithm (*Section 4.3*). To improve upon the basic algorithm, we open up the "black box" of computing MECs. We again maintain vertex sets that are candidates for good end-components such that we can guarantee that each good end-component is fully contained in one of the candidate sets. However, the maintained sets are not necessarily end-components but only preserve the property that they induce sub-MDPs, i.e., they do not have outgoing random edges. The set of candidates is again initialized with the MECs of the input MDP, but the relaxed invariant allows us to make progress in each iteration by solely computing SCCs (not MECs) and removing "bad vertices". This algorithm can be seen as "interleaving" the detection of bad vertices and the recomputations of MECs.

Improving the running time for dense (*Section 4.4*) *and sparse* (*Section 4.5*) *MDPs.* We then improve the running time using two techniques that enable us to check faster for strong connectivity after the removal of vertices.

For dense MDPs, i.e., in particular MDPs with $m \in \omega(n^{4/3}/\sqrt[3]{\log n})$, we use a sparsification technique called *Hierarchical Graph Decomposition* that was introduced by [HKW99] for edge deletions in undirected graphs and extended to vertex deletions in directed graphs and game graphs by [CH14] (see [HKL15], Section 5.3 and

Chapter 7 for further applications of this technique in our subsequent work). In contrast to [CH14], in our algorithm for Streett objectives the vertices charged in the running time argument are not removed from the MDP; we therefore refine the technique with a parallel search on the MDP based on the reverse graph.

For sparse MDPs, i.e., with $m \in O(n^{4/3}/\sqrt[3]{\log n})$, we extend the approach of "parallel local searches" of [HT96] from graphs to MDPs (see also [CJH03; CH12] and our subsequent work in [Che+17] for similar approaches).

4.2 Preliminaries and Basic Algorithm

Good end-components. The definition of good end-components, and their special case of good components for graphs, is given in Section 2.6. For a Streett objective the following is an equivalent definition.

Definition 4.2.1 (Good Streett end-component)**.** *Given an MDP P and a set* TP $= \{(L_j, U_j) \mid 1 \leq j \leq k\}$ *of target pairs, a* good (Streett) end-component *is an end-component X of P such that for each $1 \leq j \leq k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$.*

The algorithms for graphs are based on finding good components and then determining which vertices can reach these good components. For MDPs we find good end-components and then output the almost-sure winning set for a reachability objective with the union of all good end-components as target set. The correctness of this approach is shown in Section 2.6.2 (see also [BK08, Chap. 10.6.3]).

Bad vertices. Let X be a good end-component. Then $X \cap U_j = \emptyset$ implies $X \cap L_j = \emptyset$. Thus if $S \cap U_j = \emptyset$ for some vertex set S and some index j, then we have $L_j \subseteq V \setminus X$ for each end-component $X \subseteq S$. For an index j with $S \cap U_j = \emptyset$ we call the vertices of $S \cap L_j$ *bad vertices*. All algorithms in this chapter maintain candidates for good end-components and repeatedly remove the bad vertices and their random attractor from the maintained sets. In Lemma 2.6.10 we show that (a) every end-component that is contained in a set S is a subset of one of the SCCs of the subgraph induced by S and (b) that, within any sub-MDP that contains an end-component X, the set X does not intersect with any random attractor of vertices not in X. Together with the following corollary this shows the correctness of the above operations.

Corollary 4.2.2 (of Lemma 2.6.10)**.** *Given an MDP P, let X be a* good *Streett end-component with $X \subseteq S$ for some $S \subseteq V$. For each j with $S \cap U_j = \emptyset$ it holds that $X \subseteq S \setminus Attr_R(P[S], L_j \cap S)$.*

Subroutines. We use ALLMECs to denote the algorithm that computes all MECs of an MDP and MEC to denote its running time; we have MEC $= O(\min(n^2, m^{1.5}))$ by [CH14] and we assume MEC $= \Omega(m)$. In Theorem 6.3.1 we show that running time improvements to computing all MECs carry over to computing the almost-sure winning set for reachability objectives (*almost-sure reachability* for short), i.e., almost-sure reachability can be computed in time MEC (a bound of $O(\min(n^2, m^{1.5}))$ for almost-sure reachability was already shown by [CJH03; CH14], thus this does not improve the running time for this case). We further use ALLSCCs to denote an algorithm that computes all SCCs of a given MDP and SMALLESTBSCC for a subroutine that determines the smallest bottom SCC; both run time $O(m)$ [Tar72]. Note that in particular with respect to vertex sets that induce strongly connected subgraphs we often talk about the *underlying* graph G of an MDP $P = (G, (V_1, V_R))$.

Basic algorithm. The basic algorithm for MDPs with Streett objectives is given in Algorithm StreettBasic. The algorithm maintains (1) a set of already identified (maximal) good end-components goodEC, which is initially empty, and (2) a set of end-components $\mathcal{X}$ that are candidates for good end-components. The set $\mathcal{X}$ is initialized with the MECs of the input MDP P. In each iteration of the while-loop of Algorithm StreettBasic we remove an end-component X from $\mathcal{X}$ and check whether it is a good end-component. For this we find sets U_j for $1 \le j \le k$ that do not intersect with X and identify vertices in $X \cap L_j$ for such a j as bad vertices B. If there are no bad vertices, then X is a good end-component and added to goodEC. Otherwise the bad vertices and their random attractor within X are removed from X (which is correct by Corollary 4.2.2). We then compute all MECs of the sub-MDP induced by the remaining vertices of X, which identifies all remaining candidate end-components among the vertices of X. The new candidates are then added to $\mathcal{X}$. If the algorithm finds good end-components, it returns the almost-sure winning set for the reachability of the union of them.

Remark 4.2.3 (Simplifications for graphs). *In graphs we compute all non-trivial SCCs instead of all MECs, which can be done in linear time [Tar72]. Also the reachability computation in the final step of the algorithm can be done in linear time. Furthermore, no random attractors have to be computed. The total running time on graphs is obtained from the running time for MDPs by setting* MEC $= O(m)$.

Proposition 4.2.4 (Running time). *Algorithm StreettBasic can be implemented to run in* $O((\text{MEC} + b)\min(n, k))$ *time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\text{MEC})$ time. Further, by Theorem 6.3.1 the almost-sure reachability computation after the while-loop can be done in $O(\text{MEC})$ time.

Let X_v denote the end-component of $\mathcal{X}$ currently containing an arbitrary, fixed vertex $v \in V$ during Algorithm StreettBasic. In each iteration of the while-loop in which X_v is considered either (a) $B = \{x \in X_v \mid \exists j \text{ s.t. } x \in L_j \text{ and } X_v \cap U_j = \emptyset\} = \emptyset$ and X_v will not be considered further or (b) the number of vertices in X_v is reduced by at least one and we have for some $1 \le j \le k$ that $X_v \cap L_j \neq \emptyset$ before the iteration of the while-loop and $X_v \cap L_j = \emptyset$ after the while-loop. Thus each

Algorithm StreettBasic: Basic algorithm for MDPs with Streett objectives

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target pairs $\text{TP} = \{(L_j, U_j) \mid 1 \le j \le k\}$
Output : $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Streett(TP)})$
1  goodEC $\leftarrow \emptyset$
2  $\mathcal{X} \leftarrow$ allMECs($P$)
3  while $\mathcal{X} \neq \emptyset$ do
4      remove some $X \in \mathcal{X}$ from $\mathcal{X}$
5      $B \leftarrow \{x \in X \mid \exists j$ s.t. $x \in L_j$ and $X \cap U_j = \emptyset\}$
6      if $B \neq \emptyset$ then
7          $X \leftarrow X \setminus Attr_R(P[X], B)$
8          $\mathcal{X} \leftarrow \mathcal{X} \cup$ allMECs($P[X]$)
9      else
10         goodEC $\leftarrow$ goodEC $\cup \{X\}$
11 return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \text{goodEC}} X)\right)$
```

vertex and each edge of the MDP P is considered in at most $O(\min(n, k))$ iterations of the while-loop.

Consider the t-th iteration of the while-loop; let X_t denote the set removed from $\mathcal{X}$ in this iteration and let $bits(X_t) = \sum_{j=1}^{k}(|L_j \cap X_t| + |U_j \cap X_t|)$. Assume that each vertex has a list of the sets L_j and U_j for $1 \le j \le k$ it belongs to. (We can generate these lists from the lists of the target pairs in $O(b)$ time at the beginning of the algorithm.) Then we can determine the vertex set B by going through all lists of the vertices in X_t in $O(|X_t| + bits(X_t))$ time, which amounts to $O((n + b)\min(n, k))$ total time over all iterations of the while-loop. The random attractor computed in line 7 is removed and not considered further, thus its computation takes $O(m)$ time over the whole algorithm (see Section 2.5.2). The computation of all MECs in $P[X_t]$ takes total time $O(\text{MEC} \cdot \min(n, k))$ over all iterations of the while-loop. Thus the whole algorithm can be implemented in $O((\text{MEC} + b)\min(n, k))$ total time. □

Proposition 4.2.5 (Soundness)**.** *Let W be the set returned by Algorithm StreettBasic. We have $W \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, Streett(\text{TP}))$.*

Proof. By Corollary 2.6.5 it is sufficient to show that every set $X \in$ goodEC is a good end-component. The algorithm explicitly checks immediately before X is added to goodEC that we have for each $1 \le j \le k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$. Thus it remains to show that X is an end-component when it is added to goodEC. Before a set is added to goodEC, the same set is contained in the set $\mathcal{X}$. We show that all sets in $\mathcal{X}$ are end-components at any point in the algorithm by induction over the iterations of the while-loop in the algorithm. Before the first iteration of the while-loop the sets $X \in \mathcal{X}$ are the maximal end-components of P. Now consider an iteration in which a set X is removed from $\mathcal{X}$ and new sets are added to $\mathcal{X}$. First, some vertices and their random attractor in the sub-MDP $P[X]$ induced by

X are removed from X. Let X' be the remaining set of vertices. By the definition of a random attractor there are no random edges from X' to the removed random attractor. Further, by the induction hypothesis there are no random edges from X to $V \setminus X$. Thus there are no random edges from X' to $V \setminus X'$. Then the algorithm adds the MECs of the sub-MDP $P[X']$ to $\mathcal{X}$. Let $\hat{X}$ be one such MEC. Since $\hat{X}$ is a MEC in $P[X']$, it is a MEC in P if and only if it has no random edges from $\hat{X}$ to $V \setminus X'$. This holds by $\hat{X} \subseteq X'$ and the properties of X' established above. □

Proposition 4.2.6 (Completeness)**.** *Let W be the set returned by Algorithm StreettBasic. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP})) \subseteq W$.*

Proof. By Proposition 2.6.9 it is sufficient to show that at the end of Algorithm StreettBasic the union of the sets in goodEC contains all good end-components of the MDP P. We show by induction that every good end-component is full contained in a set of either goodEC or $\mathcal{X}$ before and after each iteration of the while-loop in Algorithm StreettBasic; as $\mathcal{X}$ is empty at the end of the algorithm, this implies the claim.

Before the first iteration of the while-loop, the set $\mathcal{X}$ is initialized with the MECs of P, thus the induction base holds. Let X be the set of vertices removed from $\mathcal{X}$ in an iteration of the while-loop and let X^* be the union of the good end-components contained in X. Either X is added to goodEC or we have that for some indices j the set X contains vertices of L_j but not of U_j; then for these indices the sets L_j and their random attractor are removed from X. Let $\hat{X}$ be the updated set, i.e., $\hat{X} = X \setminus \mathit{Attr}_R(P[X], B)$. By Corollary 4.2.2 we still have $X^* \subseteq \hat{X}$ after this step. Then all MECs of $P[\hat{X}]$ are added to $\mathcal{X}$. Every good end-component contained in $\hat{X}$ is completely contained in one MEC of $P[\hat{X}]$, thus the claim continues to hold after the iteration of the while-loop. □

4.3 Improving Upon the Basic Algorithm

The essential observation towards faster algorithms for MDPs with Streett objectives is the following. Consider a set X in an iteration of the basic algorithm after some vertices in $\mathit{Attr}_R(P[X], B)$ were removed. We have that there are no random edges from X to the remaining vertices in the graph and further we have for each $1 \leq j \leq k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$. Thus if $P[X]$ is still strongly connected, then X is a good end-component and is added to goodEC in one of the subsequent iterations of the algorithm. Otherwise the sub-MDP $P[X]$ consists of multiple SCCs; then we have that the bottom SCCs of $P[X]$ are end-components in P but the remaining SCCs of $P[X]$ might have outgoing random edges within $P[X]$. Note, however, that we have for any good end-component $\hat{X}$ in $P[X]$ and any SCC C of $P[X]$ that either $\hat{X} \subseteq C$ or $\hat{X} \cap C = \emptyset$, simply by the fact that every good end-component is strongly connected (see also Lemma 2.6.10 (a)). Let $\hat{X} \subseteq C$ and let *rout* be the random vertices of C with edges to vertices not in C. Then the vertices in *rout* cannot intersect with $\hat{X}$ because an end-component has no outgoing random edges. Further, also the random attractor of *rout* cannot intersect with $\hat{X}$ (Lemma 2.6.10 (b)). Thus we can remove $Attr_R(P[X], rout)$ from $P[X]$ and all good end-components that are contained in $P[X]$ are also contained in the remaining sub-MDP. The set of vertices in $C \setminus Attr_R(P[X], rout)$ has no outgoing random edges. Thus if it is still strongly connected, then it is an end-component. With this observation we can avoid computing a MEC decomposition in the while-loop of the basic algorithm and instead only compute strongly connected components and random attractors, which both can be done in linear time. Note that in the improved algorithm we do not have the property that every maintained set of vertices is an end-component (as in the basic algorithm) but still none of the maintained sets has outgoing random edges (and thus these vertex sets induce sub-MDPs, see Section 2.5.2).

In this formulation the algorithm for MDPs with Streett objectives has a very similar structure to the algorithm for graphs with Streett objectives: We repeatedly remove "bad vertices" and recompute strongly connected components. The main difference is that we additionally compute random attractors. This can be seen as opening up the "black-box" use of a MEC-decomposition algorithm and interleaving it with the identification of bad vertices.

We present the new algorithmic ideas for MDPs with Streett objectives in Algorithm StreettImpr (which is only faster for large enough k). Based on this, we show 1) in Algorithm StreettSparse how to improve the running time for dense graphs with a similar technique as in the $O(n^2)$ algorithm for MEC-decomposition [CH14] and 2) in Algorithm StreettDense how to improve the running time for sparse graphs by combining the fastest algorithms for MECs [CH14] and for Streett objectives on graphs [HT96]. Together these algorithms have a faster asymptotic running time than the basic algorithm for all parameter combinations with $k \in \omega(\log n)$; the basic algorithm is faster for $k = O(1)$ and some combinations of parameters with $k = O(\log n)$ such as $k = O(\sqrt{\log n})$ and $m = O(n^{4/3})$. In contrast to graphs with Streett objectives, current techniques do not achieve an $O((m + b)k)$ time algorithm for MDPs (which is relevant for small values of k).

Let S be a set of vertices. In our improved algorithms we use the data structure $D(S)$ from [HT96] to quickly identify and remove vertices in $S \cap L_j$ for which $S \cap U_j = \emptyset$. The following lemma describes the operations and their running times for this data structure. Let $bits(S) = \sum_{j=1}^{k} \left(|S \cap L_j| + |S \cap U_j|\right)$.

Lemma 4.3.1 ([HT96]). *After a one-time preprocessing time of $O(k)$, there is a data structure $D(S)$ for a given set of vertices S that supports the following operations:*

Construct(S) *initializes the data structure $D(S)$ for S in time $O(bits(S) + |S|)$,*

Remove(S, $D(S)$, B) *removes a set $B \subseteq V$ from S and updates $D(S)$ accordingly in time $O(bits(B) + |B|)$,* Bad($D(S)$) *returns a pointer to the set* $\{x \in S \mid \exists j \text{ s.t. } x \in L_j \text{ and } S \cap U_j = \emptyset\}$ *in constant time.*

In Algorithm StreettImpr we maintain a list Q of pairs $(S, D(S))$ of vertex sets S and their data structures $D(S)$ for disjoint vertex sets S that are candidates for good end-components. To shorten the notation, we use both $S \in Q$ and $(S, D(S)) \in Q$ interchangeably. For every set $S \in Q$ we preserve the invariant that there are no random edges from S to $V \setminus S$. The list Q is initialized with the MECs of the input MDP P. In each iteration of the outer while-loop one vertex set $S \in Q$ is selected and removed from Q. In the inner while-loop the set of *bad vertices*, that is, the vertices of $S \cap L_j$ for indices j with $S \cap U_j = \emptyset$, is identified and its random attractor is removed from S and $D(S)$. Through removing the random attractor, we maintain the property that there are no random edges from S to $V \setminus S$ at this step. Thus we have that if $P[S]$ is (still) strongly connected and contains at least one edge, then $P[S]$ is a good end-component, which we identify in line 13. If $P[S]$ does not contain an edge, we do not have to consider it further. If it contains an edge but is not strongly connected, the SCCs of $P[S]$ are identified (l. 15). For each SCC C of $P[S]$ we identify its random vertices that have edges to vertices of $S \setminus C$ (l. 18) and remove their random attractor from C (l. 21 and 24). After this step the remaining subset of C is added to Q. This happens for each SCC of $P[S]$ but there is a difference between the largest SCC and the other SCCs of $P[S]$. We construct a new data structure for all but the largest SCC (l. 25) and reuse the data structure of S for the largest SCC (l. 26. For each but the largest SCC we additionally remove its vertices from the data structure of the largest SCC (l. 23). This improves the running time because we only spend time proportional to the smaller SCCs, and a vertex can be in a smaller SCC at most $O(\log n)$ times. Note that at this point of the algorithm the sub-MDP $P[C]$ is not necessarily strongly connected since vertices have been removed after the last SCC computation. However, as $P[C]$ has no edges to vertices in $V \setminus C$, we maintain the property that there are no random edges from a vertex set in Q to other vertices. When the list Q becomes empty, the algorithm terminates. The output is then the almost-sure winning set for the reachability objective of the union of the identified good end-components, which is equal to the empty set if no good end-component exists.

Remark 4.3.2 (Simplifications for graphs)**.** *The computation of all MECs and of the a.s. winning set for the reachability objective can be replaced by the corresponding linear time computations of non-trivial SCCs and reachability in graphs. Furthermore, the set rout is always empty and all random attractor computations can be omitted. This does not affect the total running time for any of the improved algorithms.*

Proposition 4.3.3 (Running time)**.** *Algorithm StreettImpr terminates in $O(mn + b \log n)$ time.*

Proof. Using the data structure of Lemma 4.3.1, the initialization phase of Algorithm StreettImpr takes $O(k+\textsc{mec}+b+n)$ time, which is in time $O(mn+b)$. Further by

Algorithm StreettImpr: New algorithm for MDPs with Streett objectives

```
Input  : an MDP $P = ((V, E), (V_1, V_R), \delta)$ and target pairs
         $\mathrm{TP} = \{(L_j, U_j) \mid 1 \le j \le k\}$
Output : $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Streett}(\mathrm{TP}))$
goodEC ← ∅
$Q \leftarrow \emptyset$
$\mathcal{X} \leftarrow$ allMECs($P$)
for $X \in \mathcal{X}$ do
    $Q \leftarrow Q \cup \{(X, \textsc{Construct}(X))\}$
while $Q \neq \emptyset$ do
    pick and remove some $(S, D(S))$ from $Q$
    while $\textsc{Bad}(D(S)) \neq \emptyset$ do
        $A \leftarrow Attr_R(P[S], \textsc{Bad}(D(S)))$
        $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), A)$
    if $P[S]$ contains at least one edge then
        if $P[S]$ is strongly connected then
            goodEC ← goodEC $\cup \{S\}$
        else
            $\mathcal{C} \leftarrow$ allSCCs($P[S]$)
            $S' \leftarrow S$
            for $C \in \mathcal{C}$ do
                $rout \leftarrow \{v \in V_R \cap C \mid \exists w \in S' \setminus C \text{ s.t. } (v, w) \in E\}$
                $A \leftarrow Attr_R(P[C], rout)$
                if $C$ is largest SCC in $\mathcal{C}$ then
                    $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), A)$
                else
                    $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), C)$
                    $C \leftarrow C \setminus A$
                    $Q \leftarrow Q \cup \{(C, \textsc{Construct}(C))\}$
            $Q \leftarrow Q \cup \{(S, D(S))\}$
return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \mathrm{goodEC}} X)\right)$
```

Theorem 6.3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\textsc{mec})$ time.

Whenever bad vertices and their random attractor are identified in lines 8–9, they are removed in line 10 and not considered further. Thus finding bad vertices takes total time $O(n)$, identifying the random attractor of bad vertices takes total time $O(m)$ (see Section 2.5.2), and by Lemma 4.3.1 removing the bad vertices and their attractor takes total time $O(m + b)$.

After the initialization of Q with the MECs of P, all vertex sets in Q induce a strongly connected sub-MDP. Let S' be the considered vertex set when line 15 is reached, let S be its smallest superset $S \supseteq S'$ that was identified as strongly connected in the algorithm (i.e. S is either a MEC of P or an SCC computed in line 15 in a previous iteration of the algorithm). We have that S' is a proper subset of S, i.e., either bad vertices were removed from S in line 10 or a non-empty set of random vertices was identified in line 18. Hence any part of P is considered in at most n iterations of the outer while-loop. This implies that we can bound the total time spent in lines 15–19 with $O(mn)$.

The calls to Remove in line 21 take total time $O(n + b)$, as the vertices in A are not considered further. It remains to bound the time for the calls to Remove and Construct in lines 21–25. Note that we avoid to make these calls for the largest of the SCCs of the sub-MDP induced by S'. Thus whenever we call Remove and Construct for an SCC C, we have $|C| \leq |S|/2$. Hence we can charge the time for Remove and Construct to the vertices of C and to $\mathit{bits}(C)$ such that every vertex v and every $\mathit{bits}(\{v\})$ is charged $O(\log n)$ times. Thus we can bound the time for lines 23–25 with $O((n + b) \log n)$. This proves the claimed running time. □

To show the soundness of Algorithm StreettImpr, we first prove the following invariant.

Invariant 4.3.4. *For every set S with $(S, D(S)) \in Q$ there are no random edges from S to $V \setminus S$.*

Lemma 4.3.5. *Invariant 4.3.4 holds throughout Algorithm StreettImpr.*

Proof. We prove the invariant by induction over the iterations of the outer while-loop. Before the first iteration of the while-loop, Q is initialized with the maximal end-components of P and thus the invariant holds. Assume the invariant holds before the beginning of an iteration of the outer while-loop and let S be the set of vertices that is removed from Q in this iteration. In the inner while-loop some vertices and their random attractor in $P[S]$ might be removed from S. Let S' be the remaining vertices. By the definition of a random attractor there are no random edges from S' to $S \setminus S'$ and thus by the induction hypothesis there are no random edges from S' to $V \setminus S'$.

If $P[S']$ is strongly connected (l. 12–13), then no set is added to Q in this iteration of the while-loop. Otherwise the SCCs $\mathcal{C}$ of $P[S']$ are considered as candidates to be added to Q. For each set $C \in \mathcal{C}$ the random vertices *rout* (l. 18) in C with edges to vertices in $S' \setminus C$ and their random attractor A (l. 19) in $P[C]$ are removed from C (l. 21 and l. 24). Let C' be the remaining vertices. We have that there are no random edges from C' to $S' \setminus C$ by the definition of *rout* and that there are no random edges from C' to $C \setminus C'$ by the definition of A. Thus there are no random edges from C' to $V \setminus C'$ for any set C' that is added to Q (l. 25 and l 26), which shows the invariant. □

Proposition 4.3.6 (Soundness)**.** *Let the set returned by Algorithm StreettImpr be denoted by W. We have $W \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP}))$.*

Proof. By Corollary 2.6.5 it is sufficient to show that every set $X \in \mathrm{goodEC}$ is a good end-component. The algorithm explicitly checks immediately before X is added to goodEC in line 13 that $G[X]$ contains at least one edge and is strongly connected. Further we have by the termination condition of the inner while-loop that for each $1 \leq j \leq k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$. Thus it remains to show that there are no random edges from X to $V \setminus X$.

Let S be the set of vertices that is removed from Q in the iteration of the outer while-loop in which X is added to goodEC. By Invariant 4.3.4 and Lemma 4.3.5 there are no random edges from S to $V \setminus S$. If S is not equal to X, then some vertices and their random attractor within $P[S]$ are removed in the inner while-loop. By the definition of a random attractor, there are no random edges from X to $S \setminus X$ and thus to $V \setminus X$. Hence we have shown that each $X \in \mathrm{goodEC}$ is a good end-component. □

To show the completeness of Algorithm StreettImpr, we first prove the following invariant.

Invariant 4.3.7. *For each good end-component X of P and some set $Y \supseteq X$ either $Y \in \mathrm{goodEC}$ or $(Y, D(Y)) \in Q$.*

Lemma 4.3.8. *Invariant 4.3.7 holds before and after each iteration of the outer while-loop in Algorithm StreettImpr.*

Proof. We show the invariant by induction over the iterations of the outer while-loop. Before the first iteration of the outer while-loop, the set Q is initialized with the MECs of P, thus the induction base holds. Let S be the set of vertices that is removed from Q in an iteration of the outer while-loop and let $\mathcal{X}_S$ be the set of good end-components contained in S. Let S' be vertices of S that remain after the inner while-loop. By Corollary 4.2.2 we have for every $X \in \mathcal{X}_S$ that $X \subseteq S'$.

Since every end-component contains an edge, $P[S']$ contains at least one edge if $\mathcal{X}_S$ is not empty. Then either S' and thus all $X \in \mathcal{X}_S$ are added to goodEC in line 13 or the SCCs $\mathcal{C}$ of $P[S']$ are computed in line 15. By Lemma 2.6.10 (a) there exists a $C \in \mathcal{C}$ such that $X \subseteq C$ for each $X \in \mathcal{X}_S$; let X and C be such that $X \subseteq C$. Let $A = \mathit{Attr}_R(P[C], \mathit{rout})$. Since X has not outgoing random edges, we have $\mathit{rout} \cap X = \emptyset$ (line 18) and thus by Lemma 2.6.10 (b) also $X \subseteq C \setminus A$.

The set $C \setminus A$ is added to Q in lines 25 or 26, hence the claim holds after the outer while-loop. □

Proposition 4.3.9 (Completeness). *Let the set returned by Algorithm StreettImpr be denoted by W. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP})) \subseteq W$.*

Proof. By Proposition 2.6.9 it is sufficient to show that at the end of the algorithm the union of the sets in goodEC contains all good end-components of the MDP P. As Q is empty at the end of the algorithm, this is implied by Invariant 4.3.7 and Lemma 4.3.8. □

4.4 Algorithm for Dense MDPs with Streett Objectives

Algorithm StreettDense combines Algorithm StreettImpr with the ideas of the MEC-algorithm for dense MDPs of [CH14]. The difference to Algorithm StreettImpr lies in the search for strongly connected components. Fix an iteration of the outer while-loop and let S be the set of vertices that selected and removed from Q. Let $S' \subseteq S$ be the set of remaining vertices after some bad vertices and their random attractor are removed from the vertex set S in the inner while-loop. To detect a good end-component, it is essential to decide whether the sub-MDP $P[S']$ is strongly connected. For this it is sufficient to identify one strongly connected component C of the sub-MDP $P[S']$: The sub-MDP is strongly connected if and only if the SCC spans the whole sub-MDP, i.e., $C = S'$. As for Algorithm StreettImpr, the correctness of the algorithm is based on maintaining (1) that the vertex sets maintained in Q do not have outgoing random edges (Invariant 4.3.4) and (2) that each good end-component is fully contained in either Q or goodC after each iteration of the outer while-loop (Invariant 4.3.7). For maintaining these invariants, it makes no difference whether we compute all SCCs of $P[S']$ or just one. Whenever $P[S']$ is not strongly connected, there exists at least one top or bottom SCC that contains at most half of the vertices of S' (see Section 2.5.1). In Algorithm StreettDense we search for such a "small" top or bottom SCC of $P[S]$ more efficiently, discussed in detail below. The search for a top SCC is done by searching for a bottom SCC in the reverse graph *RevG*. When a bottom SCC C of *RevG* (with at most half of the vertices of S by l. 22) is identified (l. 23), then it is a top SCC in G and thus there are no random edges from $S' \setminus C$ to C but there might be random edges from C to $S' \setminus C$. Thus we can simply remove the vertices of C from the set S' (l. 24), while for the set C we determine its outgoing random edges (l. 25) and remove their random attractor from C. When a bottom SCC C of G is identified (l. 27), then C does not have outgoing random edges but we have to remove the random attractor of C from S' to maintain Invariant 4.3.4. After these steps for removing outgoing random edges (that cannot be part of a good end-component, see Lemma 2.6.10), for both C and $S' \setminus C$ the remaining vertices are added to Q (l. 29 and 30), where for the set $S' \setminus C$ the existing data structure of S is reused.

The search for bottom SCCs. To search for a bottom SCC, we use the following variant of the Hierarchical Graph Decomposition [HKW99; CH14]. For $i \in \mathbb{N}$ the graph K_i of a graph K includes the first 2^i outgoing edges of each vertex. Thus K_i has $O(n \cdot 2^i)$ edges. The main observation (Lemma 4.4.2) is that we can identify each bottom SCC with at most 2^i vertices by searching for bottom SCCs of K_i that only contain vertices for which all their outgoing edges in K are also in K_i. The search is started at level $i = 1$ and then i is doubled until such a bottom SCC is found in K_i. Note that $K_i = K$ for $i \geq \log_2 n$. When a bottom SCC is identified at level i^* but not at $i^* - 1$, then, as an immediate consequence of Lemma 4.4.2, this bottom SCC has $\Omega(2^{i^*})$ vertices (Corollary 4.4.3). Further, the number of edges in the graphs from level 1 to i^* forms a geometric series. Thus the work spent in all the levels up to i^* can be bounded in terms of the number of edges in K_{i^*}, that is, the bottom SCC of size $\Omega(2^{i^*})$ is identified in $O(n \cdot 2^{i^*})$ time. By searching "in parallel" for top and bottom SCCs and charging the needed time to the identified SCC, the total running time can be bounded by $O(n^2)$. To identify only bottom SCCs of K_i for which all the outgoing edges are present in K_i, we determine the set of "blue" vertices Bl_i that have an out-degree higher than 2^i and remove vertices that can reach blue vertices before computing SCCs (l. 15–16). In the following we provide formal definitions and proofs for Algorithm StreettDense.

Let $K = (V, E)$ be a directed graph with at most one edge between each (directed) pair of vertices in $V \times V$ and with some arbitrary but fixed ordering of the outgoing edges of each vertex.

Definition 4.4.1 (Hierarchical Graph Decomposition). *For $i \in \mathbb{N}$ let K_i be the subgraph (V, E_i) of $K = (V, E)$, where E_i contains for each vertex of V its first 2^i outgoing edges in E. Let the set Bl_i denote all vertices with out-degree more than 2^i in K.*

Note that when $i \geq \log_2(\max_{v \in V} \mathit{Outdeg}(K, v))$, then $K_i = K$ and $Bl_i = \emptyset$.

Lemma 4.4.2. *We use Definition 4.4.1.*

(1) *A set $C \subseteq V \setminus Bl_i$ is a bottom SCC in K_i if and only if it is a bottom SCC in K.*

(2) *If a set $C \subseteq V$ with $|C| \leq 2^i$ is a bottom SCC in K, then $C \subseteq V \setminus Bl_i$.*

Proof. (1) By $C \subseteq V \setminus Bl_i$ the outgoing edges of the vertices in C are the same in K_i and in K. Thus we have $K_i[C] = K[C]$ and C has no outgoing edges in K_i if and only if it has no outgoing edges in K.

(2) In K all outgoing edges of each vertex of C have to go to other vertices of C. Thus each vertex of C has an out-degree of at most $|C| \leq 2^i$ in K. □

The following corollary summarizes the properties of the Hierarchical Graph Decomposition that we use in the running time analysis of Algorithm StreettDense. Note that when in Algorithm StreettDense a set $C \subseteq S \setminus Bl_i$ is a bottom SCC in $K_i[S]$, then it does not intersect with $\textsc{GraphReach}(K_i[S], Bl_i)$ and thus it is also a bottom SCC in $K_i[Z]$ for $Z = S \setminus \textsc{GraphReach}(K_i[S], Bl_i)$.

Algorithm StreettDense: Dense MDPs with Streett objectives

```
Input  : MDP $P = (G, (V_1, V_R), \delta)$ with graph $G = (V, E)$ and
         target pairs $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$
Output : $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Streett}(\mathrm{TP}))$
goodEC ← $\varnothing$
$Q \leftarrow \varnothing$
$\mathcal{X} \leftarrow$ allMECs($P$)
for $X \in \mathcal{X}$ do
    $Q \leftarrow Q \cup \{(X, \textsc{Construct}(X))\}$
while $Q \neq \varnothing$ do
    pick and remove some $(S, D(S))$ from $Q$
    while $\textsc{Bad}(D(S)) \neq \varnothing$ do
        $A \leftarrow Attr_R(P[S], \textsc{Bad}(D(S)))$
        $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), A)$
    if $P[S]$ contains at least one edge then
        for $i \leftarrow 1$ to $\lceil \log_2(|S|) \rceil$ do
            foreach $K \in \{G, RevG\}$ do
                construct $K_i[S]$
                $Bl_i \leftarrow \{v \in S \mid Outdeg(K[S], v) > 2^i\}$
                $Z \leftarrow S \setminus \textsc{GraphReach}(K_i[S], Bl_i)$
                if $Z \neq \varnothing$ then
                    $C \leftarrow \textsc{SmallestBSCC}(K_i[Z])$
                    if $C = S$ then
                        goodEC ← goodEC $\cup \{C\}$
                        continue with next iteration of while-loop
                    if $|C| \leq |S|/2$ then
                        if $K = RevG$ then                    /* top SCC */
                            $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), C)$
                            $rout \leftarrow \{v \in V_R \cap C \mid \exists u \in S$ s.t. $(v, u) \in E\}$
                            $C \leftarrow C \setminus Attr_R(P[C], rout)$
                        else                                  /* bottom SCC */
                            $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), Attr_R(P[S], C))$
                        $Q \leftarrow Q \cup \{(C, \textsc{Construct}(C))\}$
                        $Q \leftarrow Q \cup \{(S, D(S))\}$
                        continue with next iteration of while-loop
return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \mathrm{goodEC}} X)\right)$
```

Corollary 4.4.3. (1) *If there exists a bottom SCC C with $C \subseteq V \setminus Bl_i$ in K_i but C is not contained in $V \setminus Bl_{i-1}$, then $|C| > 2^{i-1}$ and thus $|C| \in \Omega(2^i)$.*

(2) *Consider the for-loop of Algorithm StreettDense for a set S as defined in the algorithm. In the for-loop either a top or bottom SCC C with $|C| \leq |S|/2$ is identified or $G[S]$ is strongly connected.*

Proof. (1) By Lemma 4.4.2 (1) the set C is a bottom SCC in K. Thus if $|C| \leq 2^{i-1}$, then $C \subseteq V \setminus Bl_{i-1}$ by Lemma 4.4.2 (2), a contradiction.

(2) If $G[S]$ is not strongly connected, then it contains at least one top and at least one bottom SCC that are disjoint from each other, and thus at least one of them contains at most $|S|/2$ vertices. Let C^* be the smallest such top or bottom SCC with $|C^*| \leq |S|/2$, assume w.l.o.g. that C^* is a bottom SCC (otherwise replace G with *RevG* and let $i \leq i^*$), and let $i^* = \lceil \log |C^*| \rceil$. Then we have by Lemma 4.4.2 (2) that $C^* \subseteq S \setminus Bl_{i^*}$ and thus either a top or bottom SCC of $G[S]$ with $|C| \leq |S|/2$ is identified in an iteration $i < i^*$ or in iteration i^* for $K = G$ the set C^* is the smallest bottom SCC in $K[Z]$ for $Z = S \setminus$ GraphReach$(K_i[S], Bl_i)$ and is thus identified in line 22 of the for-loop. □

Proposition 4.4.4 (Running time). *Algorithm StreettDense terminates in $O(n^2 + b \log n)$ time.*

Proof. Using the data structure of Lemma 4.3.1, the initialization phase of Algorithm StreettDense takes $O(\textsc{mec} + b + n)$ time, which is in $O(n^2 + b)$ [CH14]. Further by Theorem 6.3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\textsc{mec})$ time. Removing bad vertices takes total time $O(n + b)$ by Lemma 4.3.1. Whenever a random attractor is computed, its edges are removed; thus all attractor computations take total time $O(m)$ (see Section 2.5.2). Whenever Remove or Construct are called outside of the initialization phase, the vertices that are removed resp. added are either (1) vertices for which the size of the SCC containing them was at least halved (l. 24, l. 28, and l. 29) or (2) vertices that are removed (l. 10). For each vertex, case (1) can happen at most $O(\log n)$ times and case (2) at most once, thus all calls to Remove or Construct take total time $O((n + b) \log n)$ by Lemma 4.3.1.

To efficiently construct the graphs K_i and compute Bl_i for $1 \leq i < \lceil \log_2(n) \rceil$ and $K \in \{G, RevG\}$, we maintain for all vertices a list of their incoming and outgoing edges, which we update whenever we encounter obsolete entries while constructing K_i. Each entry can be removed at most once, thus this can be done in total time $O(m)$.

Let S be the set of vertices considered in an iteration of the outer while-loop and let $|S| = n'$. We will bound the running time of the algorithm over all iterations in which vertices of S are considered by $O(n'^2)$. Applied to the vertex sets the set Q is initialized with, this implies a bound of $O(n^2)$ on the running time of the overall algorithm.

The i-th iteration of the for-loop takes $O(n' \cdot 2^i)$ time because K_i contains $O(n' \cdot 2^i)$ edges and constructing K_i and Bl_i and computing reachability, SCCs, and *rout* can all be done in time linear in the number of edges. Searching in G and *RevG* only increases the running time by a factor of two. Further *all* iterations up to the i-th iteration can be executed in time $O(n' \cdot 2^i)$ as their running times form a geometric series. Note that whenever a graph is not strongly connected, it contains at least one top SCC and at least one bottom SCC and at least one of them has at most half of the vertices. Thus in some iteration i^* a top or bottom SCC with either $C = S$ or $|C| \le n'/2$ is found by Corollary 4.4.3. Since C was not found in iteration $i^* - 1$, we have $|C| = \Omega(2^{i^*})$ by Corollary 4.4.3. It follows that $n' = \Omega(2^{i^*})$, i.e., that $2^{i^*} = O(n')$.

In the case $C = S$ (l. 20) the vertices in S are not considered further by the algorithm. Thus we can bound the time for this iteration with $O(n' \cdot 2^{\log(n')}) = O(n'^2)$, and hence the total time for this case with $O(n^2)$.

It remains to bound the time for the case $|C| \le n'/2$ (l. 22). Let $|C| = n_1$ and let c be some constant such that the time spent for the search of C is bounded by $c \cdot n_1 \cdot n'$. We denote this time spent for the vertices in the set S over the whole algorithm by $f(n')$. Finally we show $f(n') = 2cn'^2$ by induction as follows:

$$\begin{aligned} f(n') &\le f(n_1) + f(n' - n_1) + cn'n_1\,, \\ &\le 2cn_1^2 + 2c(n' - n_1)^2 + cn'n_1\,, \\ &= 2cn_1^2 + 2cn'^2 - 4cn'n_1 + 2cn_1^2 + cn'n_1\,, \\ &= 2cn'^2 + 4cn_1^2 - 3cn'n_1\,, \\ &\le 2cn'^2\,, \end{aligned}$$

where the last inequality follows from $n_1 \le n'/2$. □

Proposition 4.4.5 (Soundness). *Let the set returned by Algorithm StreettDense be denoted by W. We have $W \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, Streett(\mathrm{TP}))$.*

Proof. By Corollary 2.6.5 it is sufficient to show that every set $X \in$ goodEC is a good end-component. Let C be a set of vertices added to goodEC in line 20 and let i be the corresponding iteration of the for-loop. The subgraph induced by C is strongly connected in G_i and, since it does not contain any vertices of Bl_i, also in G by Lemma 4.4.2. Therefore we have that immediately before C is added to goodEC the algorithm verified that $G[C]$ contains at least one edge (l. 11), is strongly connected (l. 18), and Bad($D(C)$) is empty (l. 8). Thus it remains to show that there C has no outgoing random edges. Since C is obtained from some set $S \in Q$ only by removing random attractors in the inner while-loop (compare the proof of Proposition 4.3.6), it is sufficient to show that Invariant 4.3.4 holds in Algorithm StreettDense, i.e., that there are no random edges from S to $V \setminus S$ for any $S \in Q$.

Before the first iteration of the while-loop, Q is initialized with the maximal end-components of P and thus the invariant holds. Assume the invariant holds before the beginning of an iteration of the outer while-loop and let S be the set of vertices that is removed from Q in this iteration. In the inner while-loop some vertices and their random attractor in $P[S]$ might be removed from S. Let S' be the remaining vertices. By the definition of a random attractor there are no random edges from S' to $S \setminus S'$ and thus by the induction hypothesis there are no random edges from S' to $V \setminus S'$.

Then either $P[S']$ is strongly connected (l. 20) and no set is added to Q in this iteration of the while-loop or either a top or a bottom SCC C of $P[S']$ is identified in some iteration of the for-loop by Corollary 4.4.3.

If C is a top SCC (l. 23), then there are no edges from $S' \setminus C$ to C and thus $S' \setminus C$ has no outgoing random edges. Hence the invariant is maintained when $S' \setminus C$ is added to Q in line 30. In line 26 the random vertices of C with edges to vertices in $S' \setminus C$ and their random attractor are removed from C. Thus the remaining vertices of C have no random edges to $V \setminus C$ and the invariant is maintained when this vertex set is added to Q in line 29.

If C is a bottom SCC (l. 27), then there are no edges from C to $S' \setminus C$; thus the invariant is maintained when C is added to Q in line 29. In line 28 the random attractor of C is removed from $S' \setminus C$ before the remaining vertex set is added to Q in line 30, hence the invariant is maintained in all cases. □

Proposition 4.4.6 (Completeness)**.** *Let the set returned by Algorithm StreettDense be denoted by W. We have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP})) \subseteq W$.*

Proof. By Proposition 2.6.9 it is sufficient to show that at the end of the algorithm the union of the sets in goodEC contains all good end-components of the MDP P. As Q is empty at the end of the algorithm, it is sufficient to show, by induction, that Invariant 4.3.7 holds before and after each iteration of the outer while-loop in Algorithm StreettDense. Recall that the invariant guarantees that each good end-component of the given MDP P is contained in some set of either goodEC or Q.

Before the first iteration of the outer while-loop, the set Q is initialized with the MECs of P, thus the induction base holds. Let S be the set of vertices that is removed from Q in an iteration of the outer while-loop and let $\mathcal{X}_S$ be the set of good end-components contained in S. Let S' be the subset of S that is not removed in the inner while-loop. By Corollary 4.2.2 we have that removing vertices in $\textsc{Bad}(D(S))$ and their random attractor in the inner while-loop does not affect the good end-components contained in S, and this can be applied inductively. Thus we have that $X \subseteq S'$ for every $X \in \mathcal{X}_S$.

Since every end-component contains an edge, $P[S']$ contains at least one edge if $\mathcal{X}_S$ is not empty (l. 11). By Corollary 4.4.3 then either S' and thus all $X \in \mathcal{X}_S$ are added to goodEC (line 20) or an SCC $C \subsetneq S'$ of $G[S']$ is identified in line 18. By Lemma 2.6.10 (a) each $X \in \mathcal{X}_S$ is either a subset of C or of $S' \setminus C$. For $X \subseteq C$ we have $\mathit{rout} \cap X = \emptyset$ (line 25) since X has no outgoing random edges and thus $X \subseteq C \setminus \mathit{Attr}_R(P[C], \mathit{rout})$ by Lemma 2.6.10 (b). For $X \subseteq S' \setminus C$ we have $X \cap C = \emptyset$ and thus $X \subseteq S' \setminus \mathit{Attr}_R(P[S'], C)$ by Lemma 2.6.10 (b). The sets $C \setminus \mathit{Attr}_R(P[C], \mathit{rout})$ and $S' \setminus \mathit{Attr}_R(P[S'], C)$ are added to Q in lines 29 and 30, hence the invariant holds after the outer while-loop. □

4.5 Algorithm for Sparse MDPs with Streett Objectives

Algorithm StreettSparse combines Algorithm StreettImpr with the ideas of the MEC-algorithm for sparse MDPs of [CH14] and the algorithm for graphs with Streett objectives of [HT96]. The difference to Algorithm StreettImpr lies, as for dense MDPs, in the search for SCCs in a sub-MDP $P[S]$ induced by a vertex set S. The algorithm is based on the following observation: Whenever for an SCC C and a vertex set A the graph induced by $C \setminus A$ is not strongly connected, then (a) there is at least one top and at least one bottom SCC in $G[C \setminus A]$ and (b) in each top SCC of $G[C \setminus A]$ some vertex has an incoming edge from a vertex of A and in each bottom SCC some vertex has an outgoing edge to a vertex of A. We label vertices that lost an incoming edge since the last SCC computation with h (for head) and vertices that lost an outgoing edge with t (for tail). If more than $\sqrt{m/\log_2 n}$ vertices are labeled, we remove all labels and compute SCCs as in Algorithm StreettImpr; this can happen at most $\sqrt{m \log_2 n}$ times. Otherwise we search for the smallest top or bottom SCC of $P[S]$ by searching in *lock-step* from all labeled vertices. *Lock-step* means that one step in each of the searches is executed before the next step of any search is conducted and all searches are stopped as soon as the first search finishes. We search for top SCCs by searching for bottom SCCs in the reverse graph. For example, Tarjan's depth-first search based SCC algorithm [Tar72] detects a bottom SCC in time proportional to the number of edges in the bottom SCC when the search is started from a vertex inside the bottom SCC. In general any graph traversal that explores all reachable edges one by one is sufficient as described by the following observation.

Observation 4.5.1. *Assume we are given a set of start vertices T that includes at least one vertex of each bottom SCC of a graph. When we start in lock-step from each vertex of T a graph traversal of the edges reachable from the respective start vertex, then the traversal that terminates first induces a bottom SCC in the graph.*

Proof. Each vertex that is not in a bottom SCC can reach some bottom SCC and thus can reach the edges of at least one bottom SCC and at least one additional edge. As a graph traversal needs as many steps as the number of edges it traverses, one of the graph traversals that is started in the bottom SCCs with the fewest numbers of edges finishes first. □

By the label invariant described below, Algorithm StreettSparse starts a search in each bottom SCC. As there are at most $\sqrt{m/\log_2 n}$ parallel searches, the time for all the lock-step searches is $O(\sqrt{m/\log_2 n})$ times the number of edges in the smallest top or bottom SCC of $P[S]$. Since each edge can be in the smallest SCC at most $O(\log n)$ times, this leads to a total running time of $O(m^{1.5}\sqrt{\log n})$. Whenever an SCC is identified, the labels of its vertices are removed. The Invariants 4.3.4 and 4.3.7 are maintained as in Algorithm StreettImpr.

Algorithm StreettSparse: Sparse MDPs with Streett objectives

```
Input  : MDP $P = (G, (V_1, V_R), \delta)$ with graph $G = (V, E)$ and
         target pairs TP $= \{(L_j, U_j) \mid 1 \leq j \leq k\}$
Output : $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Streett}(\text{TP}))$
goodEC ← ∅; $Q \leftarrow \emptyset$; $\mathcal{X} \leftarrow$ allMECs($P$)
for $X \in \mathcal{X}$ do $Q \leftarrow Q \cup \{(X, \textsc{Construct}(X))\}$
while $Q \neq \emptyset$ do
    pick and remove some $(S, D(S))$ from $Q$
    while $\textsc{Bad}(D(S)) \neq \emptyset$ do
        $A \leftarrow Attr_R(P[S], \textsc{Bad}(D(S)))$
        $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), A)$
        add label $h$ ($t$) to vertices that just lost an incoming (resp. outgoing) edge
    $H \leftarrow \{v \in S \mid h \in label(v)\}$; $T \leftarrow \{v \in S \mid t \in label(v)\}$
    if $P[S]$ contains at least one edge then
        if $|H| + |T| = 0$ then goodEC ← goodEC ∪ $\{S\}$
        else if $|H| + |T| \geq \sqrt{m/\log_2 n}$ then
            /* as in Algorithm StreettImpr plus labels */
            remove all labels from $S$
            $\mathcal{C} \leftarrow$ allSCCs($P[S]$); $S' \leftarrow S$
            for $C \in \mathcal{C}$ do
                $rout \leftarrow \{v \in V_R \cap C \mid \exists w \in S' \setminus C \text{ s.t. } (v, w) \in E\}$
                $A \leftarrow Attr_R(P[C], rout)$
                add label $h$ ($t$) to vertices with incoming (outgoing) edge from (to) $A$
                if $C$ is largest SCC in $\mathcal{C}$ then $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), A)$
                else
                    $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), C)$
                    $C \leftarrow C \setminus A$
                    $Q \leftarrow Q \cup \{(C, \textsc{Construct}(C))\}$
            $Q \leftarrow Q \cup \{D(S)\}$
        else
            Search in lock-step from each $v \in T$ in $G[S]$ and from each $v \in H$ in
            $RevG[S]$, terminate when first search has found a bottom SCC $C$
            /* as in Alg. StreettDense plus labels */
            if $C = S$ then goodEC ← goodEC ∪ $\{S\}$
            else
                remove all labels from $C$
                if $C$ is bottom SCC in $RevG[S]$ then          /* top SCC */
                    $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), C)$
                    $rout \leftarrow \{v \in V_R \cap C \mid \exists u \in S \setminus C \text{ s.t. } (v, u) \in E\}$
                    $C \leftarrow C \setminus Attr_R(P[C], rout)$
                else                                             /* bottom SCC */
                    $(S, D(S)) \leftarrow \textsc{Remove}(S, D(S), Attr_R(P[S], C))$
                add label $h$ ($t$) to vertices that just lost an incoming (outgoing) edge
                $Q \leftarrow Q \cup \{(C, \textsc{Construct}(C))\}$
                $Q \leftarrow Q \cup \{(S, D(S))\}$
return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \text{goodEC}} X)\right)$
```

Lemma 4.5.2 (Label invariant)**.** *In Algorithm StreettSparse the following invariant is maintained for every set S with $(S, D(S))$ in Q: Either (1) no vertex of S is labeled and $G[S]$ is strongly connected or (2) in each top SCC of $G[S]$ at least one vertex is labeled with h and in each bottom SCC of $G[S]$ at least one vertex is labeled with t.*

Proof. The proof is by induction over the iterations of the outer while-loop. After the initialization of Q with the MECs of P, no vertex is labeled and every set $S \in Q$ is strongly connected. Let now S denote the set that is removed from Q at the beginning of an iteration of the outer while-loop and assume the invariant holds for S.

Observation. *We have for non-empty vertex sets W and $Z = W \setminus Y$ for some $Y \subsetneq W$ that if C is a top (bottom) SCC in $G[Z]$ but has incoming (outgoing) edges in $G[W]$, then these incoming (outgoing) edges are from (to) vertices in Y. Thus when the label invariant holds for W and we label each vertex of Z with an incoming edge from Y with h and each vertex of Z with an outgoing edge to Y with t, then the invariant holds for Z.*

By this observation the invariant remains to hold for S after the inner while-loop. In the case $|H| + |T| \geq \sqrt{m/\log_2 n}$ all labels are removed from S and then each SCC C of $G[S]$ is considered separately. Note that for each C the invariant holds and thus the invariant remains to hold for the set C added to Q when the vertices in A are removed and the corresponding labels are added in line 18. In the case $|H| + |T| < \sqrt{m/\log_2 n}$ a bottom or top SCC C of $G[S]$ is identified and all labels of C are removed. The invariant holds for C and thus the invariant remains to hold for the set C added to Q when vertices are removed from C in line 33 and the corresponding labels are added in line 36. By the above observation with $W = S$ and $Y = Attr_R(P[S], C)$ the invariant also holds for the set $S \setminus Attr_R(P[S], C)$ that is added to Q after the corresponding labels are added in line 36. □

Proposition 4.5.3 (Running time)**.** *Algorithm StreettSparse takes $O(m^{1.5}\sqrt{\log n} + b \log n)$ time.*

Proof. Using the data structure of Lemma 4.3.1, the initialization phase of Algorithm StreettSparse takes time $O(\text{MEC} + b + n)$, which is in $O(m\sqrt{m} + b)$ [CH14]. Further, by Theorem 6.3.1 the almost-sure reachability computation after the outer while-loop can be done in $O(\text{MEC})$ time. Removing bad vertices takes total time $O(n + b)$ by Lemma 4.3.1. Since a label is added only when an edge is removed by the algorithm, the total time for adding and removing labels is $O(m)$. Whenever a random attractor is computed, its edges are removed; thus all attractor computations take total time $O(m)$. Note that whenever a graph is not strongly connected, it contains some top SCC and some bottom SCC and at least one of them has at most half of the vertices. Thus whenever a top or bottom SCC C with $C \subsetneq S$ is identified in line 26, then $|C| \leq |S|/2$. This implies by Lemma 4.5.2 that whenever Remove or Construct are called (after the initialization phase), the vertices that are removed resp. added are either (a) vertices for which the size of the SCC containing them was at least halved (l. 21, 23, 31, 35, and 37) or (b) vertices that are removed (l. 7 and l. 19). Case (a) can happen at most $O(\log n)$ times, thus all calls to Remove or Construct take total time $O((n + b)\log n)$ by Lemma 4.3.1.

It remains to bound the time for identifying SCCs and determining the random boundary vertices *rout* in Case 1, $|H| + |T| \geq \sqrt{m/\log_2 n}$, and Case 2, $|H| + |T| < \sqrt{m/\log_2 n}$. Since labels are added only to the endpoints of edges that are not considered further and all labels of the considered vertices are deleted when Case 1 occurs, Case 1 can happen at most $\sqrt{m \log_2 n}$ times. Thus the total time for Case 1 can be bounded by $O(m^{1.5}\sqrt{\log n})$. In Case 2 we charge the time for the $O(\sqrt{m/\log n})$ lock-step searches to the edges in the identified SCC C. With Tarjan's SCC algorithm [Tar72] a bottom SCC is identified in time proportional to the number of edges in the bottom SCC when the search is started at a vertex in the bottom SCC, which in Algorithm StreettSparse is guaranteed by Lemma 4.5.2 for both top and bottom SCCs. Since always a top or bottom SCC C with $|C| \leq |S|/2$ is identified, each edge is charged at most $O(\log n)$ times. Thus the total time for identifying SCCs in Case 2 is $O(m^{1.5}\sqrt{\log n})$. Determining the random boundary vertices *rout* in Case 2 can be charged to the edges from C to $S \setminus C$, which are then not considered further by the algorithm. Thus the total running time of the algorithm is $O(m^{1.5}\sqrt{\log n})$. □

Proposition 4.5.4 (Correctness). *Let the set returned by Algorithm StreettSparse be denoted by W. We have $W = \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Streett}(\mathrm{TP}))$.*

Proof. By Corollary 2.6.5 and Proposition 2.6.9 it is sufficient to show that the set goodEC contains exactly all (maximal) good end-components of P when the algorithm terminates. By Lemma 4.5.2 we have that whenever a vertex set is added to goodEC in line 11, it induces a strongly connected subgraph. Thus we have that immediately before a set of vertices C is added to goodEC in line 11 or line 27, it is checked that $G[C]$ contains at least one edge, is strongly connected, and that Bad($D(C)$) is empty.

Recall Invariant 4.3.4 that states that there are no random edges between S and $V \setminus S$ for any set S that is in Q and Invariant 4.3.7 that states that each good end-component is contained in a set of goodEC or Q before and after each iteration of the outer while-loop.

For the soundness of Algorithm StreettSparse it remains to show that there are no random edges from C to $V \setminus C$. When C is added to goodEC in line 11, this follows immediately from Invariant 4.3.4, for line 27 it follows from Invariant 4.3.4 and the observation that only random attractors are removed from the superset of C that is removed from Q in the iteration in which C is added to goodEC (compare the proof of Proposition 4.3.6).

Thus for soundness it remains to show Invariant 4.3.4. For completeness we have that Q is empty at the end of Algorithm StreettSparse and hence it is sufficient to show Invariant 4.3.7. We have for each iteration of the outer while-loop: The inner while-loop is the same as in Algorithms StreettImpr and StreettDense. In the case $|H| + |T| = 0$, the currently considered set of vertices is added to goodEC and no set is added to Q. If $|H| + |T| \geq \sqrt{m/\log_2 n}$, the same operations as in Algorithm StreettImpr are performed. If $|H| + |T| < \sqrt{m/\log_2 n}$, like in Algorithm StreettDense, either a top or a bottom SCC is identified and then the same operations as in Algorithm StreettDense are applied to the identified SCC and the remaining vertices. As the operations in Algorithms StreettImpr and StreettDense preserve the invariants, the same arguments apply to Algorithm StreettSparse (see the proofs of Lemmas 4.3.5 and 4.3.8 and Propositions 4.4.5 and 4.4.6). □

4.6 Conclusion

Recently, algorithms with a running time of $\approx O(\min(m^{1.5}, n^2))$ were obtained, with similar techniques, for several problems: for MEC-decomposition [CH14], for MDPs with Streett objectives (this chapter), and for the maximal induced subgraphs that are 2-edge or 2-vertex connected in directed graphs [HKL15; Che+17]. For all these problems it would be very interesting to find faster algorithms or (conditional) lower bounds. These problems seem simple enough for the existence of a linear-time algorithm but no approach that avoids $\min(\sqrt{m}, n)$ iterations in the worst-case is known. For determining the winning sets in Büchi games only an $O(n^2)$ time algorithm is known, again with similar techniques [CH14]. Is there also an $O(m^{1.5})$ time algorithm for Büchi games?

CHAPTER 5

Parity Games

5.1 Introduction

In this chapter we present two different kinds of improved algorithms for *game graphs* with *parity objectives*, called *parity games* for short: (1) an (*explicit*) algorithm with improved running time for *dense* game graphs and (2) *set-based symbolic* algorithms with an improved number of symbolic steps. The parity games problem in general is in UP ∩ coUP [Jur98]; it is one of the rare and intriguing combinatorial problems that lie in NP ∩ coNP but are not known to be solvable in polynomial time. We first discuss the significance of parity games, μ-calculus, and symbolic algorithms, followed by previous results and our contributions.

Significance of parity games. Game graphs with parity objectives are particularly important in the verification and synthesis of systems, since all ω-regular winning conditions (such as safety, reachability, liveness, fairness) as well as all linear-time temporal logic (LTL) winning conditions can be translated to parity conditions [Saf88; Saf89]. For an (even-)parity objective, every vertex is assigned a non-negative integer priority from $\{0, 1, \ldots, c - 1\}$, and a play is winning if the highest priority visited infinitely often is even (see also Section 2.3). There is a rich literature on the algorithmic study of finite-state parity games [EJ91; Bro+97; Sei96; Jur00; VJ00; JPZ08; Sch07].

Fixed-point calculus: μ-calculus. Modal μ-calculus, a fixed-point calculus for programs, introduced in the seminal work of [Koz83], is one of the most important logics in model-checking. The fixed-point calculus consists of the alternation of least and greatest fixed-point operators and the basic formulas consist of the computation of the one-step predecessors and successors of the system. It provides an exceptional balance between expressiveness and algorithmic properties. One of the key strengths of μ-calculus is that a formula in μ-calculus is by itself a succinct description of a symbolic algorithm for the evaluation of the formula (by the evaluation of the nested fixed points). Hence μ-calculus is often referred to as the "assembly language of program logics", reflecting its comprehensiveness. Moreover, due to its elegant mathematical representation, there exist many important mathematical characterizations of μ-calculus, such as in terms of completeness [Wal00]. In a fundamental result [EJ91], the equivalence of solving parity games and μ-calculus model-checking was established. Thus the analysis of parity games, or equivalently, μ-calculus model-checking, are the core algorithmic problems for many formal analysis tools.

Explicit vs. symbolic algorithms. The algorithmic analysis of parity games has received a lot of attention [EJ91; Bro+97; Sei96; Jur00; VJ00; JPZ08; Sch07], and the algorithms can be classified broadly as *explicit* algorithms, where the algorithms operate on the explicit representation of the game graph, and *implicit or symbolic* algorithms, where the algorithms only use a set of predefined operations and does not explicitly access the game graph.

Significance of parity-3 and dense game graphs. The running time improvement of our explicit algorithm is most relevant for game graphs with parity objectives with three priorities, *parity-3 games* for short, and dense game graphs. Graphs obtained as synchronous product of several components (where each component makes transitions at each step) [KP09; Cha+16d] can lead to dense graphs. Parity-3 objectives also correspond to one-pair Streett objectives. Parity-3 games arise in many applications in verification. We sketch a few of them. (A) Timed automaton games are a model for real-time systems. The analysis of such games with reachability and safety objectives (which are the dual of reachability objectives) reduces to game graphs with parity-3 objectives [Alf+03; AF07; CHP11; CP13]. (B) The synthesis of Generalized Reactivity(1) (aka GR(1)) specifications exactly require the solution of parity-3 games [Blo+10]; GR(1) specifications are standard for hardware synthesis [PPS06] and even used in the synthesis of industrial protocols [GCH13; Blo+12][1]. (C) Finally, the problem of fair simulation [HKR02] between two systems also reduces to parity-3 games [CCK12].

Significance of set-based symbolic algorithms. Symbolic algorithms are of great significance for the following reasons: (a) first, symbolic algorithms are required for large finite-state systems that can be succinctly represented implicitly (e.g., programs with Boolean variables) and symbolic algorithms are scalable, whereas explicit algorithms do not scale; and (b) second, for infinite-state systems (e.g., real-time systems modeled as timed automata, or hybrid systems, or programs

[1]A GR(1) specification expresses that if a conjunction of Büchi objectives holds, then another conjunction of Büchi objectives must also hold, and since conjunction of Büchi objectives can be reduced in linear time to a single Büchi objective, a GR(1) specification reduces to implication between two Büchi objectives, which is a parity-3 objective.

with integer domains) only symbolic algorithms are applicable, rather than explicit algorithms. Hence for the analysis of large systems or infinite-state systems symbolic algorithms are necessary.

The most significant class of symbolic algorithms for parity games are based on *set operations*, where the allowed symbolic operations are: (a) basic set operations such as union, intersection, complement, and inclusion; and (b) one step predecessor (Pre) operations (similarly, one step successor operations (Post) operations can be defined but are not needed in this thesis). Note that the basic set operations (that only involve state variables) are much cheaper as compared to the predecessor operations (that involve both variables of the current and of the next state). Thus in our analysis we will distinguish between the basic set operations and the predecessor operations. The significance of set-based symbolic algorithms is as follows:

(1) First, in several domains of the analysis of both infinite-state systems (e.g., games over timed automata or hybrid systems) as well as large finite-state systems (e.g., programs with many Boolean variables, or bounded integer variables), the desired model-checking question is specified as a μ-calculus formula with the above set operations [AHM01; Alf+03]. Thus an algorithm with the above set operations provides a symbolic algorithm that is directly applicable to the formal analysis of such systems.

(2) Second, in other domains such as in program analysis, the one-step predecessor operators are routinely used (namely, with the weakest-precondition as a predicate transformer). A symbolic algorithm based only on the above operations thus can easily be developed on top of the existing implementations. Moreover, recent work [Bey+14] shows how efficient procedures (such as constraint-based approaches using SMTs) can be used for the computation of the above operations in infinite-state games. This highlights that symbolic one-step operations can be applied to a large class of problems.

(3) Finally, if a symbolic algorithm is described with the above very basic set of operations, then any practical improvement to these operations in a particular domain would translate to a symbolic algorithm that is faster in practice for the respective domain.

Thus the problem of an efficient set-based symbolic algorithm is at the heart of formal analysis of several fundamental models, such as in program analysis and program repair, large finite-state systems, as well as games over timed automata.

Previous results. We summarize the main previous results for finite-state game graphs with parity conditions. Consider a parity game with n vertices, m edges, and c priorities (which is equivalent to μ-calculus model-checking of transitions systems with n states, m transitions, and a μ-calculus formula of alternation depth c). In the interest of concise presentation, in the following discussion, we ignore denominators in c in the running time bounds, see Section 5.2.6, and Theorems 5.4.14 and 5.5.7 for precise bounds.

Explicit algorithms. The classical algorithm for parity games requires $O(n^{c-1}m)$ time and linear space [Zie98; McN93], which was then improved to the small-progress measure algorithm that requires $O(n^{c/2}m)$ time and $O(c \cdot n)$ space [Jur00]. The small-progress measure algorithm, which is an explicit algorithm, uses an involved domain of the product of integer priorities and *lift* operations (which is a lexicographic max and min in the involved domain). The algorithm shows that the fixed point of the lift operation computes the solution of the parity game. The lift operation can be encoded with algebraic BDDs (algebraic binary decision diagrams) [BKV04] but this does not provide a set-based symbolic algorithm. Other notable explicit algorithms for parity games are as follows: (a) a strategy improvement algorithm [VJ00], which in the worst-case is exponential [Fri09]; (b) a dominion-based algorithm [JPZ08] that requires $n^{O(\sqrt{n})}$ time and a randomized $n^{O(\sqrt{n/\log n})}$ algorithm [BSV03] (both algorithms are sub-exponential, but inherently explicit algorithms); and (c) an $\approx O(n^{c/3}m)$ time algorithm [Sch07] combining the small-progress measure and the dominion-based algorithm.

Despite the importance of parity-3 games in numerous applications and several algorithmic ideas to improve the running time for general parity games [VJ00; BSV03; JPZ08; Sch07] or Büchi games [CJH03; CH14], there has been no algorithmic improvement for parity-3 games since the $O(mn)$ time algorithm in 2000 by [Jur00].

Set-based symbolic algorithms. The basic set-based symbolic algorithm (based on the direct evaluation of the nested fixed point of the μ-calculus formula) for parity games requires $O(n^c)$ symbolic operations and linear space [EL86]. In a breakthrough result [Bro+97], a new set-based symbolic algorithm was presented that requires $O(n^{c/2+1})$ symbolic operations but also requires $O(n^{c/2+1})$ space (i.e., exponential space as compared to the linear space of the basic algorithm). A simplification of the result of [Bro+97] was presented in [Sei96].

Infinite graphs. While the above algorithms are specified for finite-state graphs, the symbolic algorithms also apply to infinite-state graphs with a finite bi-simulation quotient (such as timed-games [AD94], or rectangular hybrid games [Hen95; Kop96; HMR05]), and then n represents the size of the finite quotient.

Open questions. Given the above results, many fundamental questions remained open. A major and long-standing open question is whether there is any polynomial-time algorithm (explicit or symbolic) for parity games. In relation to set-based symbolic algorithms, a few important open questions are as follows:

(1) Does there exist a set-based symbolic algorithm that beats the $O(n^c)$ symbolic operations algorithm using only linear space?

(2) Does there exist a set-based symbolic algorithm that beats the $O(n^{c/2+1})$ symbolic operations algorithm?

Table 5.1: Explicit algorithms for parity games with few priorities.

	# Priorities				
Reference	3	4	5	6	7
[Sch07]	$O(mn)$	$O(mn^{3/2})$	$O(mn^2)$	$O(mn^{7/3})$	$O(mn^{11/4})$
[Sch07] with $m = \Theta(n^2)$	$O(n^3)$	$O(n^{7/2})$	$O(n^4)$	$O(n^{13/3})$	$O(n^{19/4})$
Theorem 5.3.9	$\boldsymbol{O(n^{2.5})}$	$\boldsymbol{O(n^3)}$	$\boldsymbol{O(n^{10/3})}$	$\boldsymbol{O(n^{15/4})}$	$\boldsymbol{O(n^{65/16})}$

(3) Does there exist a set-based symbolic algorithm that requires at most a sub-exponential number of symbolic operations?

Our contributions. *Explicit algorithm.* We show that the winning set computation for parity games with three priorities can be achieved in $O(n^{2.5})$ time. Our algorithm is faster for $m \in \omega(n^{1.5})$, and breaks the long-standing $O(mn)$ barrier for dense graphs. Our algorithm for parity-3 games also extends to general parity games and improves the running time for dense graphs when the number of priorities is sub-polynomial. Let, as in [Sch07], $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c, and let $\beta(c) = \gamma(c)/(\lfloor c/2 \rfloor + 1)$. We obtain that the running time of our algorithm is $O(n^{1+\gamma(c+1)}) = O(n^{2+\gamma(c)-\beta(c)})$ for parity games with c priorities. For a constant number of priorities this replaces m of [Sch07] by $n^{2-\beta(c)}$. Since the value of $\beta(c)$ quickly approaches 2/3 with increasing c, we have $n^{2-\beta(c)} \approx n^{4/3}$, i.e., a running time of roughly $O(n^{\gamma(c)+4/3})$. For small c we compare our running times with the Big-step algorithm of [Sch07] in Table 5.1.

Set-based symbolic algorithms. We present answers to the fundamental open questions related to set-based symbolic algorithms for parity games. Our results for game graphs with n vertices and parity objectives with c priorities are as follows:

(1) First, we present a set-based symbolic algorithm that requires $O(n^{c/2+1})$ symbolic operations and linear space. Thus it matches the symbolic operations bound of [Bro+97; Sei96] and the space requirement of the classical algorithm.

(2) Second, developing on our first algorithm, we present a set-based symbolic algorithm that requires $O(n^{\gamma(c)+1}$, i.e., roughly $O(n^{c/3+1})$, symbolic operations and linear space. Thus it improves the symbolic operations of [Bro+97; Sei96] while having the same space requirement as the classical algorithm. We also present a modification of our algorithm that requires $n^{O(\sqrt{n})}$ symbolic operations and at most linear space. Thus we also present the first (deterministic, linear-space) set-based symbolic algorithm that requires at most a sub-exponential number of symbolic operations.

In the results above the number of symbolic operations mentioned is the number of predecessor operations, and in all cases the number of required basic set operations

Table 5.2: Set-based symbolic algorithms for parity games.

Reference	Symbolic Operations	Space
[EL86; Zie98]	$O(n^c)$	$O(n)$
[Bro+97; Sei96]	$O(n^{c/2+1})$	$O(n^{c/2+1})$
Thm. 5.4.14	$\boldsymbol{O(n^{c/2+1})}$	$\boldsymbol{O(n)}$
Thm. 5.5.7	$\mathbf{min}\{\boldsymbol{n^{O(\sqrt{n})}, O(n^{c/3+1})^*}\}$	$\boldsymbol{O(n)}$

* simplified bound

(which are usually cheaper) is at most a factor of $O(n)$ more. Our main results and comparison with previous set-based symbolic algorithms are presented in Table 5.2.

Technical contributions. *Explicit algorithm.* The classical algorithm for parity-3 games repeatedly solves Büchi games such that the union of the winning sets of player 2 in the Büchi games gives the winning set for the parity-3 objective. Schewe [Sch07] showed that an algorithm for parity games by Jurdziński [Jur00] can be used to compute small subsets of the winning set of player 2, called *dominions* (see Section 2.6.3), and thereby improved the running time for general parity games. However, his ideas do not improve the running time for parity-3 games. With this algorithm dominions with at most h vertices in Büchi games can be found in time $O(mh)$. We extend this approach by using the *hierarchical graph decomposition* technique to find small dominions quickly and call the $O(n^2)$ Büchi game algorithm of [CH14] for large dominions. This extension is possible as we are able to show that, rather surprisingly, it is sufficient to consider game graphs with $O(nh)$ edges to detect dominions of size h. The hierarchical graph decomposition technique was developed by Henzinger et al. [HKW99] to handle *edge deletions* in *undirected* graphs. In [CH14] it was extended to deal with *vertex deletions* in *directed* and *game* graphs (see also Section 4.1).

Set-based symbolic algorithms. We provide a symbolic version of the progress measure algorithm. The main challenge is to succinctly encode the numerical domain of the progress measure as sets. More precisely, the challenge is to represent $\Theta(n^{c/2})$ many numerical values with $O(n)$ many sets, such that they can still be efficiently processed by a set-based symbolic algorithm. For the sake of efficiency our algorithms considers sets S_r storing all vertices with progress measure at least r. However, there are $\Theta(n^{c/2})$ many such sets S_r and thus, to reduce the space requirements to a linear number of sets, we use a succinct representation that encodes all the sets S_r with just $O(n)$ many sets, such that we can restore a set S_r efficiently whenever it is processed by the algorithm.

5.2 Preliminaries and Existing Algorithms

5.2.1 Parity Games

Let for all $c \in \mathbb{N}$ denote the set $\{0, 1, \dots, c - 1\}$ by $[c]$. A *parity game* $\mathcal{P} = (\mathcal{G}, \alpha)$ with c priorities consists of a *game graph* $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ and a *priority function* $\alpha : V \to [c]$ that assigns an integer from the set $[c]$ to each vertex. We denote the two players by $\mathcal{E}$ (for even) and $\mathcal{O}$ (for odd). Player $\mathcal{E}$ (resp. player $\mathcal{O}$) wins a play if the *highest* priority occurring infinitely often in the play is even (resp. odd). We say that the vertices in $V_{\mathcal{E}}$ are $\mathcal{E}$-vertices and the vertices in $V_{\mathcal{O}}$ are $\mathcal{O}$-vertices. We use z to denote one of the players $\{\mathcal{E}, \mathcal{O}\}$ and $\overline{z}$ to denote her opponent. *Parity-3 games* are parity games with $c = 3$ and *Büchi games* have $c = 2$. We denote the set of vertices with priority i by P_i and assume for $i \geq 0$ w.l.o.g. $|P_i| > 0$ (see Section 2.3).

One-pair Streett and parity-3 objectives. A one-pair Streett objective with pair (L_1, U_1) is equivalent to a parity game with three priorities. Let the vertices in U_1 have priority 2, let the vertices in $L_1 \setminus U_1$ have priority 1 and let the remaining vertices have priority 0. Then player 1 wins the game with the one-pair Streett objective if and only if player $\mathcal{E}$ wins the parity-3 game. In this chapter we use the notation of parity games (i.e., player $\mathcal{E}$ and player $\mathcal{O}$ instead of player 1 and player 2).

Memoryless winning strategies. See Chapter 2 for the definitions of plays, strategies, and winning sets. Since it is well-known that for parity games it is sufficient to consider memoryless strategies, in this chapter every strategy is memoryless.

Theorem 5.2.1 ([EJ91; McN93]). *For every parity game the vertices can be partitioned into the winning set $W_{\mathcal{E}}$ of player $\mathcal{E}$ and the winning set $W_{\mathcal{O}}$ of player $\mathcal{O}$. There exists a memoryless winning strategy for player $\mathcal{E}$ (resp. $\mathcal{O}$) for all vertices in $W_{\mathcal{E}}$ (resp. $W_{\mathcal{O}}$).*

5.2.2 Equivalence to Model Checking of μ-Calculus

Parity games are equivalent to modal μ-calculus; for detailed syntax and semantics we refer to [Koz83; Bro+97; EL86]; we briefly describe the main principle of the logic. Modal μ-calculus is a very expressive program logic, where the basic formulas are Boolean combination of atomic propositions, variables, and one-step predecessor operators (the existential predecessor asks for the existence of an edge, and the universal predecessor asks for all edges), and then we have the least (μ) and greatest (ν) fixed point operators. Modal μ-calculus formulas are interpreted over Kripke structures (models of transition systems with states labeled by atomic propositions, i.e., for our purpose, graphs with labels on the vertices). Modal μ-calculus model checking is linearly equivalent to solving parity games [EJ91], where the alternation depth (number of alternations of the least and greatest fixed point operators) correspond to the number of priorities of the parity game.

5.2.3 Set-based Symbolic Operations

Symbolic algorithms operate on sets of vertices, which are usually described by Binary Decision Diagrams (BDD) [Lee59; Jr78]. For the symbolic algorithms for parity games we consider the most basic form of symbolic operations, namely, *set-based symbolic* operations. More precisely, for the symbolic algorithms we only allow the following operations:

(1) *Basic set operations.* First, we allow basic set operations like $\cup$, $\cap$, $\setminus$, $\subseteq$, and $=$.

(2) *One-step operations.* Second, we allow the following one-step symbolic operations: (a) the one-step predecessor operator

$$\mathsf{Pre}(B) = \{v \in V \mid \exists u \in B : (v, u) \in E\};$$

and (b) the one-step *controllable* predecessor operator

$$\begin{aligned} \mathsf{CPre}_z(B) &= \left\{v \in V_z \mid \mathit{Out}(v) \cap B \neq \emptyset\right\} \\ &\cup \left\{v \in V_{\bar{z}} \mid \mathit{Out}(v) \subseteq B\right\}; \end{aligned}$$

i.e., the CPre_z operator computes all vertices from which z can ensure that in the next step the successor belongs to the set B. Moreover, the CPre_z operator can be defined using the Pre operator and basic set operations as follows:

$$\mathsf{CPre}_z(B) = \mathsf{Pre}(B) \setminus (V_{\bar{z}} \cap \mathsf{Pre}(V \setminus B)).$$

Algorithms that use only the above operations are called *set-based symbolic* algorithms. Additionally, successor operations can be allowed but are not needed for our algorithms. The above symbolic operations correspond to primitive operations in standard symbolic packages like CUDD [Som15].

Distinction of symbolic operations and space requirement. We clarify the distinction between the two types of symbolic operations, and also specify the space requirement of symbolic algorithms.

(1) Typically, the basic set operations are cheaper (as they encode relationships between state variables) as compared to the one-step symbolic operations (which encode the transitions and thus the relationship between the variables of the present and of the next state). Thus in our analysis we distinguish between these two types of operations.

(2) For the *space* requirements of set-based symbolic algorithms, as per standard convention, we consider that a set is stored in constant space (e.g., a set can be represented symbolically as one BDD [Bry86]). We thus consider the space requirement of a symbolic algorithm to be the maximal number of sets that the algorithm has to store.

5.2.4 μ-Calculus Based Algorithms

Note that the basic formulas in μ-calculus correspond directly to the set-based symbolic operations. Thus a μ-calculus formula by itself provides a set-based symbolic algorithm by the evaluation of the fixed points. The classical set-based symbolic algorithm [EL86] uses $O(n^c)$ symbolic operations and linear space, for μ-calculus formulas of alternation depth c. The result of [Bro+97] shows how to memorize intermediate computations to speed up the set-based symbolic computation. It reduces the number of symbolic operations to $O(n^{c/2+1})$ but to remember intermediate computations, it requires $O(n^{c/2+1})$ space.

Theorem 5.2.2. *The μ-calculus model checking problem for formulas of alternation depth c on Kripke structures of n states, and therefore parity games with c priorities and n vertices, can be solved (a) in linear space with $O(n^c)$ set-based symbolic operations [EL86]; and (b) in $O(n^{c/2+1})$ space with $O(n^{c/2+1})$ set-based symbolic operations [Bro+97].*

5.2.5 Attractors, Closed Sets, and Dominions.

We briefly recall the most relevant basic algorithmic concepts from Chapter 2.

A *player-z attractor* $A = Attr_z(\mathcal{G}, U)$ of a set of vertices $U \subseteq V$ is the maximal set of vertices from which player z can ensure to reach the set U against any strategy of player $\overline{z}$, using a memoryless strategy that does not leave A. An attractor A can be computed in time proportional to the number of incoming edges in A with an explicit algorithm and with at most $|A \setminus U| + 1$ set-based symbolic operations.

A set of vertices $U \subseteq V$ is *z-closed* if all outgoing edges of z-vertices in U lead to other vertices of U and all $\overline{z}$-vertices in U have at least one outgoing edge to a vertex of U. Player $\overline{z}$ has a memoryless strategy to keep a play within a z-closed set U against all strategies of player z. Recall that the complement of a z-attractor is z-closed (Lemma 2.5.1 (1)).

A (non-empty) set of vertices $D \subseteq V$ is a *z-dominion* if player z has a winning strategy from every vertex of D that does not leave D. Every z- dominion is $\overline{z}$-closed. By Lemma 2.6.11 (3) we can determine winning sets in an iterative manner by removing dominions and their attractor and recursing on the remaining game graph.

By a slight abuse of notation, we denote the sub-game induced by a z-closed set U by $(\mathcal{G}[U], \alpha)$, where the priority function α is evaluated only on U and we say that the highest priority of $(\mathcal{G}[U], \alpha)$ is $\max_{v \in U} \alpha(v)$. In this case we further interpret a set P_i for a priority i w.r.t. the subgame, i.e., as $P_i \cap U$.

5.2.6 Algorithms for Parity Games

In this section we present the key existing algorithms for parity games along with the main ideas for correctness.

5.2.6.1 Classical Algorithm

In the following we describe a classical algorithm for parity games by [Zie98; McN93] and provide intuition for its correctness. Our symbolic big-step algorithm presented in Section 5.5 uses the same overall structure as the classical algorithm but determines dominions using our symbolic progress measure algorithm presented in Section 5.4.

Algorithm ClassicParity: Classical algorithm for parity games

```
Input  : parity game $\mathcal{P} = (\mathcal{G}, \alpha)$, with
         game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ and
         priority function $\alpha : V \to [c]$.
Output : winning sets $(W_{\mathcal{E}}, W_{\mathcal{O}})$ of player $\mathcal{E}$ and player $\mathcal{O}$
1  if $c = 1$ then return $(V, \emptyset)$
2  let $z$ be player $\mathcal{E}$ if $c$ is odd and player $\mathcal{O}$ otherwise
3  $W_{\overline{z}} \leftarrow \emptyset$
4  repeat
5      $\mathcal{G}' \leftarrow \mathcal{G} \setminus Attr_z(\mathcal{G}, P_{c-1})$
6      $(W'_{\mathcal{E}}, W'_{\mathcal{O}}) \leftarrow$ClassicParity$(\mathcal{G}', \alpha)$
7      $A \leftarrow Attr_{\overline{z}}(\mathcal{G}, W'_{\overline{z}})$
8      $W_{\overline{z}} \leftarrow W_{\overline{z}} \cup A$
9      $\mathcal{G} \leftarrow \mathcal{G} \setminus A$
10 until $W'_{\overline{z}} = \emptyset$
11 $W_z \leftarrow V \setminus W_{\overline{z}}$
12 return $(W_{\mathcal{E}}, W_{\mathcal{O}})$
```

Informal description of classical algorithm. Let z be $\mathcal{E}$ if c is odd and $\mathcal{O}$ if c is even. Let $\mathcal{G}$ be the game graph as maintained by the algorithm. The classical algorithm repeatedly identifies $\overline{z}$-dominions by recursive calls for a parity game $\mathcal{P}' = (\mathcal{G}', \alpha)$ with one priority less that is obtained by temporarily removing the z-attractor of P_{c-1}, i.e., the vertices with highest priority, from the game. In other words, the steps are as follows:

(1) Obtain the game $\mathcal{P}'$ by removing the z-attractor of P_{c-1}.

(2) If the winning set $W'_{\overline{z}}$ of player $\overline{z}$ in the parity game $\mathcal{P}'$ is non-empty, then its $\overline{z}$-attractor A is added to the winning set $W_{\overline{z}}$ of $\overline{z}$ and removed from the game graph $\mathcal{G}$. The algorithm recurses on the remaining game graph.

(3) Otherwise all vertices in the parity game $\mathcal{P}'$ are winning for player z. In this case the algorithm terminates and the remaining vertices $V \setminus W_{\overline{z}}$ are returned as the winning set of player z.

The pseudocode of the classical algorithm is given in Algorithm ClassicParity.

Key intuition for correctness. The correctness argument is inductive over the number of priorities and has the following two key aspects.

(1) The winning set of player $\overline{z}$ in $\mathcal{P}'$ is a $\overline{z}$-dominion in $(\mathcal{G}, \alpha)$ because the vertices in $\mathcal{P}'$ are z-closed by Lemma 2.5.1 (1). Thus the attractor of the winning set of player $\overline{z}$ in $\mathcal{P}'$ can be removed as part of the winning set of player $\overline{z}$ and it suffices to solve the remaining game by Lemma 2.5.1 (3).

(2) If the algorithm terminates in some iteration where all vertices in $\mathcal{P}'$ are winning for z, then a winning strategy for player z on the remaining game can be constructed by combining her winning strategy in the subgame $\mathcal{P}'$ (by the inductive hypothesis over the number of priorities as $\mathcal{P}'$ has a strictly smaller number of priorities) with her attractor strategy to the vertices with highest priority, and the fact that the set of remaining vertices $V \setminus W_{\overline{z}}$ is $\overline{z}$-closed by Lemma 2.5.1 (1).

The classical algorithm can be interpreted both as an explicit algorithm as well as a set-based symbolic algorithm, since it only uses attractor computations and set operations. The following theorem summarizes the results for the classical algorithm for parity games.

Theorem 5.2.3. *[Zie98; McN93] Algorithm ClassicParity correctly computes the winning sets of parity games; as an explicit algorithm it requires $O(n^{c-1} \cdot m)$ time and linear space; and as a set-based symbolic algorithm it requires $O(n^c)$ symbolic one step and set operations, and linear space.*

5.2.6.2 Progress Measure Algorithm

We first provide basic intuition for the progress measure and then provide the formal definitions. By the results of [Jur00] the task of solving parity games can be reduced to computing the progress measure. In Section 5.4 we present a set-based symbolic algorithm to compute the progress measure.

High-level intuition: progress measure. Towards a high-level intuition behind the progress measure, consider an $\mathcal{E}$-dominion D, i.e., player $\mathcal{E}$ wins on all vertices of D without leaving D. Fix a play started at a vertex $u \in D$ in which player $\mathcal{E}$ follows her winning strategy on D. In the play from some point on the highest priority visited by the play, say α^*, has to be even. Let v_* be the vertex after which the highest visited priority is α^* (recall that memoryless strategies are sufficient for parity games). Before v_* is visited, the play might have visited vertices with odd priority higher than α^* but the number of these vertices has to be less than n. The *progress measure* is based on a so-called *lexicographic ranking* function that assigns a *rank* to each vertex v, where the rank is a "vector of counters" for the number of times player $\mathcal{O}$ can force a play to visit an odd priority vertex before a vertex with higher even priority is reached.

- If player $\mathcal{O}$ can ensure a counter value of at least n, then she can ensure that a cycle with highest priority odd is reached from v and therefore player $\mathcal{E}$ cannot win from the vertex v.
- Conversely, if player $\mathcal{O}$ can reach a cycle with highest priority odd before reaching a higher even priority, then she can also force a play to visit an odd priority n times (thus a counter value of n) before reaching a higher even priority.

In other words, a vertex u is in the $\mathcal{E}$-dominion D if and only if player $\mathcal{O}$ *cannot* force any counter value to reach n. When a vertex u is classified as winning for player $\mathcal{O}$, it is marked with the rank $\top$ and whenever $\mathcal{O}$ has a strategy for some vertex v to reach a $\top$-ranked vertex, it is also winning for player $\mathcal{O}$ and thus ranked $\top$. The progress measure itself is computed by updating the rank of a vertex according to the ranks of its successors, which corresponds to computing the least simultaneous fixed point for all vertices with respect to "ranking functions".

Strategies from progress measure. An additional property of the progress measure is that the ranks assigned to the vertices of the $\mathcal{E}$-dominion provide a certificate for a winning strategy of player $\mathcal{E}$ within the dominion, namely, player $\mathcal{E}$ can follow edges that lead to vertices with "lower or equal" rank with respect to a specific ordering of the ranks.

We next provide formal definitions of *rank*, the *ranking function*, the ordering on the ranks, the *lift*-operators, and finally the *progress measure* (see also [Jur00]).

The progress measure domain $M^\infty_{\mathcal{G}}$. We consider parity games with n vertices and highest priority $c-1$. Let n_i be the number of vertices with priority i for odd i (i.e., $n_i = |P_i|$), let $n_i = 0$ for even i, and let $N_i = [n_i + 1]$ for $0 \le i < c$. Let $M_{\mathcal{G}} = (\{0\} \times N_1 \times \{0\} \times N_3 \times \ldots \times N_{c-2} \times \{0\})$ for odd c and $M_{\mathcal{G}} = (\{0\} \times N_1 \times \{0\} \times N_3 \times \ldots \times N_{c-1})$ for even c be the product domain where every even index is 0 and every odd index i is a number between 0 and n_i. The *progress measure domain* is $M^\infty_{\mathcal{G}} = M_{\mathcal{G}} \cup \{\top\}$, where $\top$ is a special element called the top element. Then we have $|M^\infty_{\mathcal{G}}| = 1 + \prod_{i=1}^{\lfloor c/2 \rfloor}(n_{2i-1} + 1) = O\big(\big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ [Jur00] (this bound uses the w.l.o.g. assumption that for each priority > 0 there is at least one vertex with the respective priority).

Ranking functions. A *ranking function* $\rho : V \to M^\infty_{\mathcal{G}}$ assigns to each vertex a *rank* r that is either one of the c dimensional vectors in $M_{\mathcal{G}}$ or the top element $\top$.

Note that a rank has at most $\lfloor c/2 \rfloor$ non-zero entries. Informally, we call the entries of a rank with an odd index i a "counter" because as long as the top element is not reached, it counts (with "carry", i.e., if n_i is reached, the next highest counter is increased by one and the counter at index i is reset to zero) the number of times a vertex of priority i is reached before a vertex of higher priority is reached (from some specific start vertex). The co-domain of ρ is $M_{\mathcal{G}}^{\infty} = M_{\mathcal{G}} \cup \{\top\}$ and we index the elements of the vectors from 0 to $c - 1$.

Lexicographic comparison operator <. We use the following ordering $<$ of the ranks assigned by ρ: the vectors are considered in the lexicographical order, where the left most entry is the least significant one and the right most entry is the most significant one, and $\top$ is the maximum element of the ordering. We write $\bar{0}$ to refer to the all zero vector (i.e., the minimal element of the ordering) and $\bar{N}$ to refer to the maximal vector $(n_0, n_1, \dots, n_{c-1})$ (i.e., the second largest element, after $\top$, in the ordering).

Lexicographic increment and decrement operations. Given a rank r, i.e., either a vector or $\top$, we refer to the successor in the ordering $<$ by $\mathrm{inc}(r)$ (with $\mathrm{inc}(\top) = \top$), and to the predecessor in the ordering $<$ by $\mathrm{dec}(r)$ (with $\mathrm{dec}(\bar{0}) = \bar{0}$). We also consider restrictions of inc and dec to fewer dimensions, which are described below. Given a vector $x = (x_0, x_1, x_2, \dots, x_{c-1})$, we denote by $\langle x \rangle_\ell$ (for $0 \le \ell < c$) the vector $(0, 0, \dots, 0, x_\ell, \dots, x_{c-1})$, where we set all elements with index less than ℓ to 0; in particular $x = \langle x \rangle_0$. Intuitively, we use the notation $\langle x \rangle_\ell$ to "reset the counters" for priorities lower than ℓ when a vertex of priority ℓ is reached (as long as we have not counted up to the top element). Moreover, we also generalize the ordering to a family of orderings $<_\ell$ where $x <_\ell y$ for two vectors x and y iff $\langle x \rangle_\ell < \langle y \rangle_\ell$; the top element $\top$ is the maximum element of each ordering. In particular, $x <_0 y$ iff $x < y$ and in our setting also $x <_1 y$ iff $x < y$. We further have restricted versions inc_ℓ and dec_ℓ of inc and dec; note that dec_ℓ is a partial function and that ℓ will be the priority of the vertex v for which we want to update its rank and x will be the rank of one of its neighbors in the game graph.

$$\mathrm{inc}_\ell(x) = \begin{cases} \langle x \rangle_\ell & \text{if } \ell \text{ is even and } x < \top\,, \\ \min\{y \in M_{\mathcal{G}}^{\infty} \mid y >_\ell x\} & \text{if } \ell \text{ is odd and } x < \top\,, \\ \top & \text{if } x \text{ is } \top\,, \end{cases}$$

and

$$\mathrm{dec}_\ell(x) = \begin{cases} \bar{0} & \text{if } x \text{ is } \bar{0}\,, \\ \min\{y \in M_{\mathcal{G}} \mid x = \mathrm{inc}_\ell(y)\} & \text{otherwise}\,. \end{cases}$$

For $\bar{0} < x < \top$ we have $\mathrm{inc}_\ell(\mathrm{dec}_\ell(x)) = \mathrm{dec}_\ell(\mathrm{inc}_\ell(x)) = \langle x \rangle_\ell$ while for $\top$ we only have $\mathrm{inc}_\ell(\mathrm{dec}_\ell(\top)) = \top$ and for $\bar{0}$ only $\mathrm{dec}_\ell(\mathrm{inc}_\ell(0)) = 0$. By the restriction of inc by the priority ℓ of v, for both even and odd priorities the counters for lower (odd) priorities are reset to zero as long as the top element is not reached. For an odd ℓ

additionally the counter for ℓ is increased or, if the counter for ℓ has already been at n_ℓ, then one of the higher counters is increased while the counter for ℓ is reset to zero as well; if no higher counter can be increased any more, then the rank of v is set to $\top$.

The best **operation.** Recall the interpretation of the progress measure as a witness for a player-$\mathcal{E}$ winning strategy on an $\mathcal{E}$-dominion, where player $\mathcal{E}$ wants to follow a path of non-increasing rank. The function best we define next reflects the ability of player $\mathcal{E}$ to choose the edge leading to the lowest rank when he owns the vertex, while for player-$\mathcal{O}$ vertices all edges need to lead to non-increasing ranks if player $\mathcal{E}$ can win from this vertex. The function best for each vertex v and ranking function ρ is given by

$$\operatorname{best}(\rho, v) = \begin{cases} \min\{\rho(w) \mid (v,w) \in E\} & \text{if } v \in V_{\mathcal{E}}\,, \\ \max\{\rho(w) \mid (v,w) \in E\} & \text{if } v \in V_{\mathcal{O}}\,. \end{cases}$$

The Lift **operation and the progress measure.** Finally, the lift operation implements the incrementing of the rank of a vertex v according to its priority and the ranks of its neighbors:

$$\mathtt{Lift}(\rho, v)(u) = \begin{cases} \operatorname{inc}_{\alpha(v)}(\operatorname{best}(\rho, v)) & \text{if } u = v\,, \\ \rho(u) & \text{otherwise}\,. \end{cases}$$

The $\mathtt{Lift}(.,v)$-operators are monotone and the *progress measure* for a parity game is defined as the *least simultaneous fixed point of all* $\mathtt{Lift}(.,v)$*-operators.* The progress measure can be computed by starting with the ranking function equal to the all-zero function and iteratively applying the $\mathtt{Lift}(.,v)$-operators in an arbitrary order [Jur00]. See [Jur00] for a worst-case example for any lifting algorithm. By the following result, the winning set of player $\mathcal{E}$ can be obtained from the progress measure by selecting those vertices whose rank is a vector, i.e., smaller than $\top$.

Lemma 5.2.4. *[Jur00] For a given parity game and the progress measure ρ with codomain $M^\infty_{\mathcal{G}}$, the vertices with $\rho(v) < \top$ are exactly the winning vertices for player $\mathcal{E}$.*

This implies that to solve parity games it is sufficient to provide an algorithm that computes the least simultaneous fixed point of all $\mathtt{Lift}(.,v)$-operators. The $\mathtt{Lift}$ operation can be computed explicitly in $O(m)$ time, which gives the SmallProgressMeasure algorithm of [Jur00].

Theorem 5.2.5. *[Jur00] Algorithm SmallProgressMeasure correctly computes the winning sets of parity games and it is an explicit algorithm that requires $O(m \cdot |M^\infty_{\mathcal{G}}|) = O\big(m \cdot \big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ time and $O(n \cdot c)$ space.*

5.2.6.3 Sub-exponential and Big-step Algorithms

Finally, we discuss the sub-exponential and the big-step algorithm for parity games.

Sub-exponential algorithm [JPZ08]. The sub-exponential time algorithm of [JPZ08] is based on the following modification of the classical algorithm. Before the recursive call, which finds a non-empty dominion, the algorithm enumeratively and explicitly searches for all dominions of size at most $\sqrt{n}$; if it succeeds to find a dominion, then its attractor is removed from the game; otherwise, the subsequent recursive call is guaranteed to find a dominion of size $> \sqrt{n}$. A clever analysis of the recurrence relation shows that the running time of the algorithm is at most $n^{O(\sqrt{n})}$, yielding the first deterministic sub-exponential time algorithm for parity games. However, the algorithm is inherently explicit and enumerative (it enumerates with a brute-force search all dominions of size at most $\sqrt{n}$). We refer the above algorithm as SubExp algorithm.

Theorem 5.2.6. *[JPZ08] Algorithm SubExp correctly computes the winning sets of parity games, and it is an explicit algorithm that requires $n^{O(\sqrt{n})}$ time and linear space.*

Big-step algorithm. The progress measure algorithm and the sub-exponential algorithm were combined in [Sch07] to obtain the big-step algorithm. The main idea is to use the progress measure to identify (small) dominions of size $\leq h + 1$, for some given integer $h \in [1, n-1]$. Given that an $\mathcal{E}$-dominion is of size $\leq h+1$, player $\mathcal{E}$ must have a strategy from each vertex of the $\mathcal{E}$-dominion to reach a vertex with an even priority by visiting at most h vertices with odd priorities. Thus, one considers a product domain $M_h \subseteq M_{\mathcal{G}}$ containing only the vectors of $M_{\mathcal{G}}$ whose elements sum up to at most h. The co-domain M_h^∞ of the ranking function ρ is then given by $M_h^\infty = M_h \cup \{\top\}$ and the function $\mathrm{inc}(r)$ and $\mathrm{dec}(r)$ are then only defined on the restricted domain M_h^∞ (the min in the definitions is over M_h^∞ instead of $M_{\mathcal{G}}^\infty$). Again the corresponding progress measure for a parity game is defined as the least simultaneous fixed point of all $\mathtt{Lift}(., v)$-operators. The identification of $\mathcal{E}$-dominions from the progress measure is achieved by selecting those vertices whose rank is a vector, i.e., smaller than $\top$.

Lemma 5.2.7. *[Sch07] For a given parity game with n vertices and the progress measure ρ with co-domain M_h^∞ for some integer $h \in [1, n-1]$, the set of vertices $\{v \in V \mid \rho(v) < \top\}$ is an $\mathcal{E}$-dominion that contains all $\mathcal{E}$-dominions with at most $h+1$ vertices.*

In our explicit algorithm we use the SmallProgressMeasure algorithm with the co-domain M_h^∞ as a subroutine to compute z-dominions with at most $h+1$ vertices.

Remark 5.2.8. *Note that the progress measure algorithm determines $\mathcal{E}$-dominions. We can compute $\mathcal{O}$-dominions (including a winning strategy within the dominion) by adding one to each priority and changing the roles of the two players. If c has been even before this modification, this does not increase the bound on the number of symbolic operations for the dominion search because the number of odd priorities, and therefore the possible number of non-empty indices of a rank vector, does not increase.*

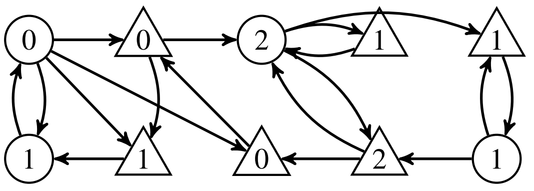

Figure 5.1: An example of a parity game $\mathcal{P} = (\mathcal{G}, \alpha)$ with three priorities. Circles denote $\mathcal{E}$-vertices, triangles denote $\mathcal{O}$-vertices. The values in the vertices are the priorities.

Lemma 5.2.9 ([Jur00; Sch07]). *Let $(\mathcal{G}, \alpha)$ be a parity game with game graph $\mathcal{G}$ with n vertices and m edges, a priority function α, and c priorities. There is an explicit algorithm* ProgressMeasure$(\mathcal{G}, \alpha, h, z)$ *that returns the union of all z-dominions of size at most $h+1$, including a winning strategy for z, in time $O(c \cdot m \cdot \binom{h+\lceil c/2 \rceil}{h})$.*

Combining the sub-exponential algorithm with the progress measure algorithm to identify small dominions gives the BigStep algorithm for parity games.

Theorem 5.2.10. *[Sch07] Let $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c. Algorithm BigStep correctly computes the winning sets of parity games and it is an explicit algorithm that requires $O\big(m \cdot \big(\frac{\kappa \cdot n}{c}\big)^{\gamma(c)}\big)$ time for some constant κ and $O(n \cdot c)$ space.*

In [Sch17] (in press, personal communication) the running time bound for the BigStep algorithm is improved further to $O\big(m\big(\frac{6e^{5/3}n}{c^2}\big)^{\gamma(c)}\big)$.

5.3 Explicit Algorithm for Dense Game Graphs

In this section we present our explicit algorithm that computes the winning sets of both players for a parity game $\mathcal{P} = (\mathcal{G}, \alpha)$ with $c \leq \sqrt{n}$ priorities in time $O(n^{1+\gamma(c+1)})$, where $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c, and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c. The special case for $c = 3$ is described explicitly in the conference version of this section [CHL15].

Initialization (steps 1–3 of Procedure Parity**).** If all vertices of the parity game have priority zero, player $\mathcal{E}$ wins from all vertices and thus the algorithm terminates and returns the set of all vertices as the winning set of player $\mathcal{E}$ and the empty set as winning set of player $\mathcal{O}$. Otherwise we set z according to the parity of the highest priority $c-1$ in the parity game. The procedure then iteratively determines the winning set of player $\overline{z}$ and in the end identifies W_z as the complement of $W_{\overline{z}}$.

Iterated vertex deletions (steps 4–12 of Procedure Parity**).** The algorithm repeatedly removes vertices from the game graph $\mathcal{G}$. During the algorithm, we

Procedure PARITY($\mathcal{G}, \alpha, h$)

```
Input  : game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ with $n = |V|$,
         priority function $\alpha : V \to [c]$ with $c \geq 1$, and
         parameter $h \in [1, n] \cap \mathbb{N}$
Output : winning sets $(W_{\mathcal{E}}, W_{\mathcal{O}})$ of player $\mathcal{E}$ and player $\mathcal{O}$
1  if $c = 1$ then return $(V, \emptyset)$
2  let $z$ be player $\mathcal{E}$ if $c$ is odd and player $\mathcal{O}$ otherwise
3  $W_{\overline{z}} \leftarrow \emptyset$
4  repeat
5      $W'_{\overline{z}} \leftarrow$ DOMINION($\mathcal{G}, \alpha, h, \overline{z}$)
6      if $W'_{\overline{z}} = \emptyset$ and $c \geq 3$ then
7          $\mathcal{G}' \leftarrow \mathcal{G} \setminus Attr_z(\mathcal{G}, P_{c-1})$
8          $(W'_{\mathcal{E}}, W'_{\mathcal{O}}) \leftarrow$ PARITY($\mathcal{G}', \alpha$)
9      $A \leftarrow Attr_{\overline{z}}(\mathcal{G}, W'_{\overline{z}})$
10     $W_{\overline{z}} \leftarrow W_{\overline{z}} \cup A$
11     $\mathcal{G} \leftarrow \mathcal{G} \setminus A$
12 until $W'_{\overline{z}} = \emptyset$
13 $W_z \leftarrow V \setminus W_{\overline{z}}$
14 return $(W_{\mathcal{E}}, W_{\mathcal{O}})$
```

denote by $\mathcal{G}$ the remaining game graph after vertex deletions. The vertex removal is achieved by identifying parts of the winning set of player $\overline{z}$, i.e, $\overline{z}$-dominions, and removing their attractors.

Dominion find and attractor removal. The algorithm repeatedly finds dominions of player $\overline{z}$ in parity games $\mathcal{P}'$ where the highest priority is at most $c - 2$. The parity game $\mathcal{P}'$ is constructed by removing the z-attractor of vertices with priority $c - 1$ from $\mathcal{G}$ (this is implicit in the Procedure DOMINION and explicit before the recursive call to Procedure PARITY; more details follow). After a dominion in the parity game $\mathcal{P}'$ is found, its $\overline{z}$-attractor is removed from $\mathcal{G}$. Then the search for $\overline{z}$-dominions is continued on the remaining vertices. If all vertices in the parity game $\mathcal{P}'$ are winning for z, i.e., no $\overline{z}$-dominion exists in $\mathcal{P}'$, then the procedure terminates. The winning set of player $\overline{z}$ is the union of the $\overline{z}$-attractors of all found $\overline{z}$-dominions. The remaining vertices are winning for player z. We now describe the steps to find $\overline{z}$-dominions.

Steps of dominion find. For the search for $\overline{z}$-dominions in the parity game $\mathcal{P}'$ we use two different procedures, Procedure DOMINION and a recursive call to Procedure PARITY. We first search for "small" $\overline{z}$-dominions with up to h vertices with Procedure DOMINION, where h is a parameter that will be set later to balance the running times of the two procedures. If no $\overline{z}$-dominion is found, we can conclude that either all $\overline{z}$-dominions contain more than h vertices or the winning set of $\overline{z}$

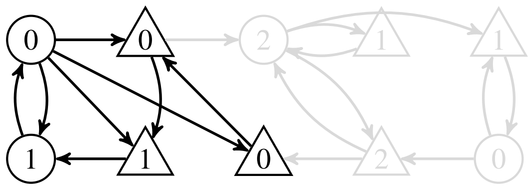

Figure 5.2: The resulting Büchi game $(\mathcal{G}', \alpha)$ (in black) after removing $Attr_{\mathcal{E}}(\mathcal{G}, P_2)$ from $\mathcal{G}$. The vertex in the top left corner is an $\mathcal{E}$-vertex with out-degree larger than $\sqrt{n}$, i.e., a blue vertex of $\mathcal{G}_i'$ for $i \le \log_2(\sqrt{n})$. The two Büchi vertices in the bottom left corner are contained in $Attr_{\mathcal{E}}(\mathcal{G}_i', Bl_i)$ and we have $\mathcal{D}_i = \emptyset$.

Procedure DOMINION($\mathcal{G}$, α, h, $\overline{z}$)

```
Input  : game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$,
         priority function $\alpha : V \to [c]$ with $c \ge 2$,
         parameter $h \in [1, n] \cap \mathbb{N}$, and
         player $\overline{z}$
Output : a $\overline{z}$-dominion that contains all $\overline{z}$-dominions with at most $h$ vertices or
         possibly the empty set if no such $\overline{z}$-dominion exists
1  for $i \leftarrow 1$ to $\lceil \log_2(h) \rceil$ do
2      construct $\mathcal{G}_i$
3      $Bl_i \leftarrow \{v \in V_{\overline{z}} \mid Outdeg(\mathcal{G}_i, v) = 0\} \cup \{v \in V_z \mid Outdeg(\mathcal{G}, v) > 2^i\}$
4      $\mathcal{G}_i' \leftarrow \mathcal{G}_i \setminus Attr_z(\mathcal{G}_i, P_{c-1} \cup Bl_i)$
5      if $c = 2$ then
6          $D_i \leftarrow$ the vertices of $\mathcal{G}_i'$
7      else
8          $D_i \leftarrow$ PROGRESSMEASURE($\mathcal{G}_i', \alpha, 2^i, \overline{z}$)
9      if $D_i \ne \emptyset$ then
10         return $D_i$
11 return $\emptyset$
```

on the current game graph is empty. In the latter case the algorithm terminates. The former case occurs at most n/h times and in such a case we use the recursive call to Procedure PARITY on the parity game $\mathcal{P}'$ with one priority less to obtain a $\overline{z}$-dominion. Below we describe the details of Procedure DOMINION.

Example 5.3.1 (Illustration of the algorithm for parity-3.). *Figure 5.1 shows a parity game with priorities* $\{0, 1, 2\}$*, and Figure 5.2 shows the Büchi game we obtain when we remove the $\mathcal{E}$-attractor of the vertices with the highest priority* 2*. Figure 5.3 shows the winning set of player $\mathcal{O}$ in the Büchi game, which is an $\mathcal{O}$-dominion in the parity game, and its $\mathcal{O}$-attractor in the original game graph. Finally, Figure 5.4 shows the remaining game graph after the removal of the $\mathcal{O}$-dominion and its attractor.*

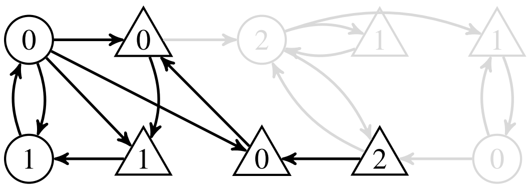

Figure 5.3: The winning set of player $\mathcal{O}$ in the Büchi game $(\mathcal{G}', \alpha)$ and its $\mathcal{O}$-attractor in $\mathcal{G}$.

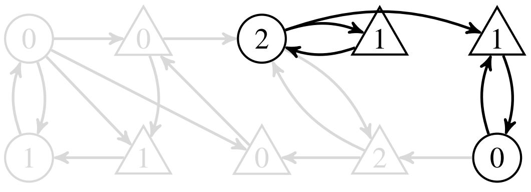

Figure 5.4: The remaining graph. The two vertices on the right are an $\mathcal{O}$-dominion, the remaining vertices form the winning set of player $\mathcal{E}$.

Graph decomposition for Procedure Dominion. In the Procedure Dominion we use the following variant of the hierarchical graph decomposition. For a game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ we denote its decomposition with respect to player $\overline{z}$ by $\{\mathcal{G}_i\}$. We consider the incoming edges of each vertex in a fixed order: First the edges from vertices of $V_{\overline{z}}$, then the remaining edges. We define $\lceil \log_2 n \rceil$ graphs $\mathcal{G}_i = (V, E_i)$ for *levels* $1 \leq i \leq \lceil \log_2 n \rceil$ where the set of edges E_i contains for each vertex $v \in V$ with $Outdeg(\mathcal{G}, v) \leq 2^i$ all its outgoing edges in E and in addition for each vertex $v \in V$ its first 2^i incoming edges in E. Note that (1) $E_i \subseteq E_{i+1}$, (2) $|E_i| \leq 2^{i+1} n$, and (3) $\mathcal{G}_{\lceil \log_2 n \rceil} = \mathcal{G}$. We color z-vertices v with $Outdeg(\mathcal{G}, v) > 2^i$ and $\overline{z}$-vertices without outgoing edges in $\mathcal{G}_i$ *blue* for $\mathcal{G}_i$ and denote the set of blue vertices by Bl_i. All other vertices are called *white.*

Procedure Dominion. The Procedure Dominion searches for $\overline{z}$-dominions in parity games with highest priority at most $c - 2$, where the game graphs are given by parts of the subgraphs $\mathcal{G}_i$ that only contain white vertices. The search for $\overline{z}$-dominions is started at $i = 1$. As long as no $\overline{z}$-dominion is found, the index i is increased one by one up to at most $i = \lceil \log_2(h) \rceil$. To (a) ensure that $\overline{z}$-dominions found in the subgraph $\mathcal{G}_i$ are also $\overline{z}$-dominions in $\mathcal{G}$ and (b) reduce the number of priorities in the parity game by one, the z-attractor of the vertices in $Bl_i \cup P_{c-1}$ is removed from $\mathcal{G}_i$. Then for $c \geq 3$ the ProgressMeasure algorithm (Lemma 5.2.9) is used to find $\overline{z}$-dominions of size at most $O(2^i)$ in the subgame induced by the remaining vertices of $\mathcal{G}_i$. For $c = 2$, i.e., Büchi games, we have that any non-empty set of remaining vertices is a dominion of player $\overline{z}$ (in this case the algorithm is equivalent to the one of [CH14]).

In the remaining part of this section we prove the correctness and running time of Procedure Parity (including the calls to Procedure Dominion). The following lemma captures the essence of why the hierarchical graph decomposition is helpful for graph games. The lemma is a generalization of related lemmata for Büchi games in [CH14] and for parity-3 games in the conference version of this section [CHL15]. The first part of the lemma is essential for the correctness of the hierarchical graph decomposition technique on game graphs: It shows that every z-closed set in $\mathcal{G}_i$ that consists only of white vertices is also z-closed in $\mathcal{G}$. The second part is essential for the running time: Every z-closed set in $\mathcal{G}$ that, including its $\overline{z}$-attractor, contains at most h vertices, is a z-closed set induced by white vertices in $\mathcal{G}_{\lceil \log_2 h \rceil}$. Note that the $\overline{z}$-attractor of a z-closed set is itself z-closed and thus this holds in particular for maximal z-closed sets of size at most h.

Lemma 5.3.2. *Let $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ be a game graph and $\{\mathcal{G}_i\}$ its hierarchical graph decomposition w.r.t. to player $\overline{z}$. For $1 \le i \le \lceil \log_2 n \rceil$ let Bl_i be the set consisting of the player-$\overline{z}$ vertices that have no outgoing edge in $\mathcal{G}_i$ and the player-z vertices with more than 2^i outgoing edges in $\mathcal{G}$.*

(1) *If a set $S \subseteq V \setminus Bl_i$ is z-closed in $\mathcal{G}_i$, then S is z-closed in $\mathcal{G}$.*

(2) *If a set $S \subseteq V$ is z-closed in $\mathcal{G}$ and $|Attr_{\overline{z}}(\mathcal{G}, S)| \le 2^i$, then (i) $\mathcal{G}_i[S] = \mathcal{G}[S]$, (ii) the set S is in $V \setminus Bl_i$, and (iii) S is z-closed in $\mathcal{G}_i$.*

Proof. (1) By $S \subseteq V \setminus Bl_i$ we have for all $v \in S \cap V_z$ that $Out(\mathcal{G}, v) = Out(\mathcal{G}_i, v)$. Thus if $Out(\mathcal{G}_i, v) \subseteq S$, then also $Out(\mathcal{G}, v) \subseteq S$. Each edge of G_i is contained in G, thus we have for all $v \in S \cap V_{\overline{z}}$ that $Out(G_i, v) \cap S \neq \emptyset$ implies $Out(G, v) \cap S \neq \emptyset$.

(2) Since S is closed for player z and $|S| \le 2^i$, (a) the set S does not contain vertices $v \in V_z$ with $Outdeg(\mathcal{G}, v) > 2^i$. Furthermore, for every vertex of S also the vertices in $V_{\overline{z}}$ from which it has incoming edges are contained in $Attr_{\overline{z}}(\mathcal{G}, S)$. Thus by $|Attr_{\overline{z}}(\mathcal{G}, S)| \le 2^i$ no vertex of S has more than 2^i incoming edges from vertices of $V_{\overline{z}}$. Hence, by the ordering of incoming edges in the construction of $\mathcal{G}_i$, we obtain (b) for the vertices of S all incoming edges from vertices of $V_{\overline{z}}$ are contained in E_i. Combining (a), i.e., $Out(\mathcal{G}, v) = Out(\mathcal{G}_i, v)$ for $v \in S \cap V_z$, and (b), i.e., $(u, w) \in E_i$ for $u \in V_{\overline{z}}$ and $w \in S$, we have (i) $\mathcal{G}_i[S] = \mathcal{G}[S]$. Since S is closed for player z in $\mathcal{G}$, every vertex $u \in S \cap V_{\overline{z}}$ has an outgoing edge to another vertex $w \in S$ in $\mathcal{G}$. Thus in particular these edges (u, w) are contained in E_i and hence every vertex $u \in S \cap V_{\overline{z}}$ has an outgoing edge to another vertex $w \in S$ in $\mathcal{G}_i$. It follows from (a) that (ii) $S \cap Bl_i = \emptyset$, and from $E_i \subseteq E$ that (iii) S is closed for player z in $\mathcal{G}_i$. □

In the next lemma we consider not just z-closed sets but $\overline{z}$-dominions and thus argue additionally about winning strategies of player $\overline{z}$. The first part of the lemma shows the soundness of Procedure Dominion, while the second part shows completeness and is crucial for the running time analysis of the overall algorithm.

Lemma 5.3.3. *Let the parity game $(\mathcal{G}, \alpha)$ with c priorities, the parameter h, and the player $\overline{z}$ be the input to Procedure* Dominion. *Let D_i, $\mathcal{G}_i'$ and Bl_i be as in Procedure* Dominion.

(1) *Every $D_i \neq \emptyset$ is a $\overline{z}$-dominion in the parity game $(\mathcal{G}, \alpha)$ with $D_i \cap P_{c-1} = \emptyset$.*

(2) *If there exists a $\overline{z}$-dominion D with $|Attr_{\overline{z}}(\mathcal{G}, D)| \leq 2^i$ in $(\mathcal{G}, \alpha)$ such that $D \cap P_{c-1} = \emptyset$, then D is a $\overline{z}$-dominion in the parity game $(\mathcal{G}_i', \alpha)$.*

Proof. (1) By definition the highest priority in $\mathcal{G}_i'$ and thus in D_i is at most $c-2$. Let V_i' be the vertices of $\mathcal{G}_i'$. By Lemma 2.5.1 (1) the set V_i' is z-closed in $\mathcal{G}_i$ and by $V_i' \cap Bl_i = \emptyset$ and Lemma 5.3.2 (1) also in $\mathcal{G}$. If $c = 2$, we have $V_i' = D_i$ and by $D_i \cap P_{c-1} = \emptyset$ that D_i is a $\overline{z}$-dominion in $(\mathcal{G}, \alpha)$. If $c \geq 3$, then D_i is the set returned by ProgressMeasure and thus by Lemma 5.2.9 a $\overline{z}$-dominion in the parity game $(\mathcal{G}_i', \alpha)$. Since V_i' is z-closed in $\mathcal{G}_i$, the set D_i is also a $\overline{z}$-dominion in $\mathcal{G}_i$ by Lemma 2.5.1 (1). With $D_i \cap Bl_i = \emptyset$ we have (a) the set D_i is z-closed in $\mathcal{G}$ by Lemma 5.3.2 (1) and (b) all outgoing edges of vertices of V_z are present in $\mathcal{G}_i$. Thus by $E_i \subseteq E$ the winning strategy of player $\overline{z}$ for the vertices of D_i in $\mathcal{G}_i$ is also a winning strategy in $\mathcal{G}$ and hence D_i is a $\overline{z}$-dominion in the parity game $(\mathcal{G}, \alpha)$.

(2) By Lemma 5.3.2 (2) we have (i) $\mathcal{G}[D] = \mathcal{G}_i[D]$, (ii) $D \cap Bl_i = \emptyset$, and (iii) D is z-closed in $\mathcal{G}_i$. Thus (a) D is contained in $\mathcal{G}_i'$ and (b) player $\overline{z}$ can play the same winning strategy in $\mathcal{G}_i[D]$ as in $\mathcal{G}[D]$. □

In the following corollary we state the insights of the previous two lemmata as needed for the running time analysis. The first part shows that when we use the hierarchical graph decomposition with increasing level i to search for a $\overline{z}$-dominion and we have to go up to level i^*, then the found $\overline{z}$-dominion, or at least its $\overline{z}$-attractor (which is again a $\overline{z}$-dominion), contains a number of vertices proportional to 2^{i^*}, which allows us to charge the work done in the search to the vertices in the identified dominion. The second part of the corollary states that no "small" $\overline{z}$-dominions exist in the maintained parity game if Procedure Dominion returns the empty set, where "small" is specified by the parameter h that will be set to balance the running time of Procedure Dominion and the recursive calls. In this case either no $\overline{z}$-dominion exists in the parity game and the algorithm terminates or the subsequent recursive call identifies a $\overline{z}$-dominion with more than h vertices; the latter can happen at most $O(n/h)$ times and allows us to bound the number of iterations in Procedure Parity. For the proof of the second part we use that every $\overline{z}$-dominion D contains a subset D' that is a $\overline{z}$-dominion itself and does not contain any vertex with priority $c-1$. Intuitively, D' is the set of vertices that are contained in the cycles of D that are induced by the memoryless winning strategy of player $\overline{z}$ in $\mathcal{G}[D]$.

Lemma 5.3.4 (Proposition 2 of [Jur00]). *Let $\mathcal{P}$ be a parity game with c priorities and let z be $\mathcal{E}$ if c is odd and $\mathcal{O}$ otherwise. Let D be any $\overline{z}$-dominion in $\mathcal{P}$. Then there exists a $\overline{z}$-dominion $D' \subseteq D$ with $D' \cap P_{c-1} = \emptyset$.*

Corollary 5.3.5. *Let the parity game $(\mathcal{G}, \alpha)$ with c priorities, the parameter h, and the player $\overline{z}$ be the input to Procedure* Dominion. *Let D_i, $\mathcal{G}_i'$ and Bl_i be as in Procedure* Dominion.

(1) *If for some $i > 1$ we have $D_i \neq \emptyset$ but $D_{i-1} = \emptyset$, then $|Attr_{\overline{z}}(\mathcal{G}, D_i)| > 2^{i-1}$.*

(2) *If Procedure* Dominion *returns the empty set, then we have for every $\overline{z}$-dominion D in the given parity game $|Attr_{\overline{z}}(\mathcal{G}, D)| > h$.*

Proof. (1) By Lemma 5.3.3 (1) we have that D_i is a $\overline{z}$-dominion in $(\mathcal{G}, \alpha)$ with $D_i \cap P_{c-1} = \emptyset$. Assume by contradiction that $|Attr_{\overline{z}}(\mathcal{G}, D_i)| \leq 2^{i-1}$. Then by Lemma 5.3.3 (2) the set D_{i-1} is not empty, a contradiction.

(2) Assume by contradiction that Procedure Dominion returns the empty set and there exists a $\overline{z}$-dominion D with $|Attr_{\overline{z}}(\mathcal{G}, D)| \leq h$ in $(\mathcal{G}, \alpha)$. By Lemma 5.3.4 in this case also a $\overline{z}$-dominion $D' \subseteq D$ with $D' \cap P_{c-1} = \emptyset$ exists. By Lemma 5.3.3 (2) the set D' is a $\overline{z}$-dominion in the parity game $(\mathcal{G}_i', \alpha)$ for $i \geq \lceil \log_2(|D'|) \rceil$ and thus in particular for $i = \lceil \log_2(h) \rceil$. Hence by Lemma 5.2.9 Procedure Dominion returns a non-empty set, a contradiction. □

Combining the running time bound for ProgressMeasure by Lemma 5.2.9 with Corollary 5.3.5, we bound the running time of Procedure Parity without the recursive calls.

Lemma 5.3.6. *Let $(\mathcal{G}, \alpha)$ be a parity game with a game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ with $n = |V|$, a priority function α, and c priorities and let $h \in [1, n]$ be a parameter with $h = n$ for $c = 2$. The running time of Procedure* Parity$(\mathcal{G}, \alpha)$ without the recursive calls, *and without the attractor computation before the recursive calls, is $O\big(c \cdot n^2 \cdot \binom{h+\lfloor c/2 \rfloor}{h}\big)$ for $c \geq 3$, $O(n^2)$ for $c = 2$, and $O(n)$ for $c = 1$.*

Proof. All operations before and after the repeat-until loop can be done in $O(n)$ time, which shows $O(n)$ for $c = 1$. Further the attractor computations and the updates of the maintained sets in lines 9–11 can be done in total time $O(m)$. Thus it remains to bound the total time for the calls to Procedure Dominion.

To efficiently construct the graphs $\mathcal{G}_i$ and the vertex sets Bl_i, we maintain ordered lists of the incoming and outgoing edges of each vertex. These lists can be updated whenever an obsolete entry is encountered in the construction of $\mathcal{G}_i$; as each entry is removed at most once, this takes total time $O(m)$.

We now analyze the time spent in an iteration i of the for-loop in Procedure Dominion. The graph $\mathcal{G}_i$ contains $O(2^i \cdot n)$ edges and both $\mathcal{G}_i$ and Bl_i can be constructed from the maintained lists of in- and outedges in $O(2^i \cdot n)$ time. Also the attractor computation takes time $O(2^i \cdot n)$. Thus for $c = 2$ the time in iteration i is $O(2^i \cdot n)$, while for $c \geq 3$ the time is dominated by the call to ProgressMeasure. Note that with the attractor computation the vertices with the highest priority are removed from the parity game, thus the call to ProgressMeasure is done for a parity game with $c - 1$ priorities and parameter $h = 2^i$. Hence by Lemma 5.2.9 iteration i for $c \geq 3$ takes time

$$O\left(c \cdot n \cdot 2^i \cdot \binom{2^i + \lceil (c-1)/2 \rceil}{2^i}\right) = O\left(c \cdot n \cdot 2^i \cdot \binom{2^i + \lfloor c/2 \rfloor}{2^i}\right).$$

The time for all iterations up to the i-th iteration forms a geometric series and thus satisfies the same running time bound.

Let i^* be the last iteration of the for-loop in a call to Procedure Dominion. Let z be $\mathcal{E}$ if c is odd and $\mathcal{O}$ otherwise. By Corollary 5.3.5 either (1) $\boldsymbol{D}_{i^*}$ is a $\overline{z}$-dominion with $|Attr_{\overline{z}}(\mathcal{G}, \boldsymbol{D}_{i^*})| > 2^{i^*-1}$ vertices or (2) $i^* = \lceil \log_2(h) \rceil$ and $\mathcal{G}$ does not contain any $\overline{z}$-dominion $\boldsymbol{D}$ with $Attr_{\overline{z}}(\mathcal{G}, \boldsymbol{D}) \leq h$ vertices. In case (2) either (2a) $c \geq 3$ and a $\overline{z}$-dominion with more than h vertices in its $\overline{z}$-attractor is detected in the subsequent recursive call to Procedure Parity or (2b) there is no $\overline{z}$-dominion in the maintained parity game and this is the last iteration of the repeat-until loop in the Procedure Parity. Case (2b) can happen at most once and its running time is bounded by $O(n^2)$ for $c = 2$ and by

$$O\left(c \cdot n \cdot 2^{\log_2(h)} \cdot \binom{2^{\log_2(h)} + \lfloor c/2 \rfloor}{2^{\log_2(h)}}\right)$$

for $c \geq 3$, which can be bounded by $O\big(c \cdot n^2 \cdot \binom{h+\lfloor c/2 \rfloor}{h}\big)$. In the cases (1) and (2a) more than 2^{i^*-1} vertices are removed from the maintained graph in this iteration. We charge each of these vertices $O\big(c \cdot n \cdot \binom{2^i+\lfloor c/2 \rfloor}{2^i}\big)$ time, which can be bounded by $O\big(c \cdot n \cdot \binom{h+\lfloor c/2 \rfloor}{h}\big)$ (per vertex). Hence the total running time is bounded by

$$O\left(c \cdot n^2 \cdot \binom{h + \lfloor c/2 \rfloor}{h}\right). \qquad \square$$

To bound the running time including the recursive calls, we use a similar analysis as for the BigStep algorithm in [Sch07; Sch08]. The running time for parity games with c priorities of the BigStep algorithm is $O(m \cdot (\kappa n/c)^{\gamma(c)})$ for $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c, and some constant κ (Theorem 5.2.10). Further let $\beta(c) = \gamma(c)/(\lfloor c/2 \rfloor + 1)$. It can easily be verified that $\gamma(c+1) = 1 + \gamma(c) - \beta(c)$ holds. Similar to [Sch07; Sch08], we set $h = n^{\beta(c)}$ for $c \geq 3$ and additionally $h = n$ for $c = 2$. We show by induction over c a running time bound of $O(n^{1+\gamma(c+1)}) = O(n^{2+\gamma(c)-\beta(c)})$ for parity games with c colors, i.e., for a constant number of priorities we replace m by $n^{2-\beta(c)}$.

Lemma 5.3.7. *For parity games with $c \leq \sqrt{n}$ priorities Procedure* Parity *takes time $O(n^{1+\gamma(c+1)})$.*

Proof. For the base case of $c = 2$ we have $\gamma(c+1) = 1$ and no recursive calls. Thus the running time of Procedure Parity for $c = 2$ is $O(n^2)$ by Lemma 5.3.6

(in this case Procedure Parity is equivalent to the algorithm of [CH14]). Suppose Procedure Parity runs in time $O(n^{1+\gamma(c)})$ for a parity game with $c-1 \geq 2$ priorities. We show that this implies that Procedure Parity runs in time $O(n^{1+\gamma(c+1)})$ for a parity game with c priorities for $3 \leq c \leq \sqrt{n}$. Let $h = n^{\beta(c)}$ for $\beta(c) = \gamma(c)/(\lfloor c/2 \rfloor + 1)$. We have $\beta(c) \geq 1/2$ for all $c \geq 3$ and thus $h \geq \sqrt{n}$. By Lemma 5.3.6 the time spent in Procedure Parity without the recursive calls is $O\big(c \cdot n^2 \cdot \binom{h+\lfloor c/2 \rfloor}{h}\big)$. With Stirling's approximation of $(x/e)^x \leq x!$ we have

$$\begin{aligned}\binom{h+\lfloor c/2\rfloor}{h} &\leq \frac{(h+\lfloor c/2\rfloor)^{\lfloor c/2\rfloor}}{\lfloor c/2\rfloor!}\,,\\ &\leq \left(\frac{(h+\lfloor c/2\rfloor)\cdot e}{\lfloor c/2\rfloor}\right)^{\lfloor c/2\rfloor}\,.\end{aligned}$$

Using $3 \leq c \leq \sqrt{n} \leq h$, we obtain

$$\begin{aligned}\binom{h+\lfloor c/2\rfloor}{h} &\leq \left(\frac{2eh+ec}{c-1}\right)^{\lfloor c/2\rfloor}\,,\\ &\leq \left(\frac{5eh}{c}\right)^{\lfloor c/2\rfloor}\,.\end{aligned}$$

Thus we have

$$c \cdot \binom{h+\lfloor c/2\rfloor}{h} \leq \frac{(5e)^{\lfloor c/2\rfloor}}{c^{\lfloor c/2\rfloor-1}} h^{\lfloor c/2\rfloor}\,,$$

which is $\leq h^{\lfloor c/2\rfloor}$ for $c \geq (5e)^{3/2}$ and $\leq \kappa h^{\lfloor c/2\rfloor}$ for some constant κ for $c < (5e)^{3/2}$. Hence the time without the recursive calls is bounded by $O(n^2 \cdot h^{\lfloor c/2\rfloor})$, which is equal to $O(n^{2+\beta(c)\lfloor c/2\rfloor})$ for $h = n^{\beta(c)}$. By the choice of $\beta(c)$ we have $\beta(c)\lfloor c/2\rfloor = \gamma(c) - \beta(c)$ and thus we can write this bound as $O(n^{2+\gamma(c)-\beta(c)})$. By Corollary 5.3.5 there are at most $O(n/h) = O(n^{1-\beta(c)})$ recursive calls to Procedure Parity. Each recursive call is for a parity game with one priority less and thus takes time $O(n^{1+\gamma(c)})$. Hence the total time for all recursive calls is bounded by $O(n^{1-\beta(c)+1+\gamma(c)})$. For $\gamma(c)$ as defined above we have $\gamma(c+1) = 1 + \gamma(c) - \beta(c)$, which completes the proof. □

By Lemma 5.3.3 (1) every $\overline{z}$-dominion found in the parity game with one priority less on $\mathcal{G}'$ is indeed a $\overline{z}$-dominion in the original parity game on $\mathcal{G}$. Together with Lemma 2.5.1 (3) this implies that the computed set $W_{\overline{z}}$ is indeed a part of the winning set of player $\overline{z}$ in the parity game. To show the correctness of Procedure Parity, we provide a winning strategy for player z for all remaining vertices.

Lemma 5.3.8 (Correctness). *Given a parity game $\mathcal{P}$, let z be player $\mathcal{E}$ if c is odd and player $\mathcal{O}$ otherwise and let W_z and $W_{\overline{z}}$ be the output of Procedure* Parity. *We have: (1)* (Soundness). *$W_{\overline{z}} \subseteq W_{\overline{z}}(P)$; and (2)* (Completeness). *$W_{\overline{z}}(P) \subseteq W_{\overline{z}}$.*

Proof. The first part on soundness follows from Lemmata 5.3.3 (1) and 2.5.1 (3). We now prove the completeness result. When Procedure Parity terminates, the winning set $W'_{\overline{z}}$ of player $\overline{z}$ in the parity game $(\mathcal{G}', \alpha)$ is empty. Also note that since the algorithm removes attractors of $\overline{z}$, the set W_z is closed for $\overline{z}$ by Lemma 2.5.1 (1). Consider the set $Z = W_z \cap P_{c-1}$, its attractor $X = Attr_z(\mathcal{G}, Z)$, and the subgame induced by $U = W_z \setminus X$. Note that the game graphs $\mathcal{G}[U]$ and $\mathcal{G}'[U]$ coincide. Thus all vertices of U must be winning for player z in the parity game $(\mathcal{G}', \alpha)$ as otherwise $W'_{\overline{z}}$ would have been non-empty for $(\mathcal{G}', \alpha)$. We prove the lemma by describing a winning strategy for player z in $\mathcal{P}$ for all vertices in W_z. For vertices of $Z \cap V_z$ the winning strategy chooses an edge in W_z, which exists since W_z is $\overline{z}$-closed. For vertices in $X \setminus Z$ player z follows her attractor strategy to Z. In the subgame induced by $U = W_z \setminus X$ player z follows her winning strategy in the parity game $(\mathcal{G}', \alpha)$. Then in a play either (i) X is visited infinitely often; or (ii) from some point on only vertices of U are visited. In the former case, the attractor strategy ensures that then some vertex of Z with priority $c - 1$ is visited infinitely often; and in the later case, the subgame winning strategy ensures that the highest priority visited infinitely often has the same parity as $c - 1$. It follows that $W_z \subseteq W_z(P)$, i.e., $W_{\overline{z}}(P) \subseteq W_{\overline{z}}$, and the desired result follows. □

Lemmata 5.3.7 and 5.3.8 yield the following result.

Theorem 5.3.9. *Procedure* Parity *correctly computes the winning sets in parity games with n vertices and $c \leq \sqrt{n}$ priorities in $O(n^{1+\gamma(c+1)})$ time, where $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c.*

Computation of winning strategies. In parity-3 games the previous results for computing winning strategies for the players in their respective winning sets are as follows: The small-progress measure algorithm of [Jur00] requires $O(mn)$ time to compute the winning strategy of player $\mathcal{E}$ and $O(n^2 m)$ time to compute the winning strategy for player $\mathcal{O}$; Schewe [Sch08] shows how to modify the small-progress measure algorithm to compute the respective winning strategies of both players in $O(mn)$ time. Schewe's running time bound for general parity games also holds when both winning strategies are requested [Sch08]. We show that our algorithm also computes the respective winning strategies without increasing the running time, i.e., in $O(n^{2.5})$ time for parity-3 games and in $O(n^{1+\gamma(c+1)})$ time for parity games with a constant number of priorities c.

For *parity-3* we first observe that for Büchi games [CH14] we can construct in $O(n^2)$ time also the respective winning strategies of both players since the algorithm is based on identifying closed sets and attractors, and the corresponding winning strategies are identified immediately with the computation. The proof of Lemma 5.3.8 describes the strategy computation for a winning strategy of player $\mathcal{O}$ which involves an attractor strategy and the sub-game strategy for Büchi games, each of which can be computed in $O(n^2)$ time. A winning strategy for player $\mathcal{E}$ is obtained in the iterations of the algorithm, i.e., whenever we obtain a dominion by solving Büchi games we also obtain a corresponding winning strategy, and similarly for the attractor computation. Thus the winning strategy for player $\mathcal{E}$ can be computed in $O(n^{2.5})$ time.

For *general parity games* the winning strategies for both players are constructed in a similar way; the argument uses parity-3 games as base case and then induction over the recursive calls. Let z be player $\mathcal{E}$ if c is odd and player $\mathcal{O}$ otherwise. First note that the time bound in Lemma 5.2.9, and therefore the time bound of Procedure Dominion, includes the computation of a winning strategy for player $\overline{z}$ within a $\overline{z}$-dominion determined by Procedure Dominion. The winning strategy of player $\overline{z}$ is a combination of his winning strategies for the dominions identified in Procedure Dominion and the dominions identified in the recursive calls for parity games with one priority less and the corresponding attractor strategies. The winning strategy of player z, as described in Lemma 5.3.8, is identified in the last iteration of the repeat-until loop and consists of her winning strategy for the parity game for which the last recursive call is made and her attractor strategy to vertices with priority $c - 1$.

Corollary 5.3.10. *Winning strategies for player $\mathcal{E}$ and player $\mathcal{O}$ in their respective winning sets in parity games with n vertices and $c \leq \sqrt{n}$ priorities can be computed in $O(n^{1+\gamma(c+1)})$ time.*

5.4 Symbolic Progress Measure Algorithm

In this section we present a set-based symbolic algorithm for parity games, with n vertices and c priorities, by showing how to compute a progress measure using only set-based symbolic operations (see Section 5.2.3 for the definition of symbolic operations).

Key challenge. Recall that the main challenge for an efficient set-based symbolic algorithm similar to the SmallProgressMeasure algorithm is to represent $\Theta(n^{c/2})$ many numerical values succinctly with $O(n)$ many sets, such that they can still be processed efficiently by a symbolic algorithm.

Key concepts: The sets S_r and the ranking function $\rho_{\{S_r\}_r}$. Recall from Section 5.2.6.2 that the progress measure for parity games is defined as the least simultaneous fixed point of the $\mathtt{Lift}(\rho, v)$-operators on a ranking function $\rho : V \to M^{\infty}_{\mathcal{G}}$. There are two key aspects of our algorithm:

(1) *Symbolic encoding of numerical domain.* In our symbolic algorithm we cannot directly deal with the ranking function but have to use sets of vertices to encode it. We first formulate our algorithm with sets S_r for $r \in M^{\infty}_{\mathcal{G}}$ that contain all vertices that have rank r or higher; that is, given a function ρ, the corresponding sets are $S_r = \{v \mid \rho(v) \geq r\}$. On the other hand, given a family of sets $\{S_r\}_r$, the corresponding ranking function $\rho_{\{S_r\}_r}$ is given by

$\rho_{\{S_r\}_r}(v) = \max\{r \in M^{\infty}_{\mathcal{G}} \mid v \in S_r\}$. This formulation encodes the numerical domain with sets but uses exponential in c many sets.

(2) *Space efficiency.* In Section 5.4.3 we directly encode the ranks with one set corresponding to each possible index-value pair. This reduces the required number of sets to linear at the cost of increasing the number of set operations only by a factor of n; the number of one-step symbolic operations does not increase.

Organization. Our results are organized as follows. First, we present the result for parity games with 5 priorities. Second, we present the general case, where we present the set-based symbolic algorithm to compute the progress measure, which still requires exponential space, i.e., an exponential number of sets. Finally, we present the modification which reduces the space requirement to a linear number of sets, increasing the number of set operations only by a factor of n.

5.4.1 Illustration: Parity Games with 5 Priorities

In this section we informally introduce our symbolic algorithm to compute the progress measure, using parity games with 5 priorities as an important special case.

Intuition for ranks r and sets S_r. We first provide some intuition for the ranks r and the sets S_r for parity games with 5 priorities. The rank r has an index for each of the priorities $\{0, 1, 2, 3, 4\}$ but may contain non-zero entries only for the odd priorities $\{1, 3\}$. Thus, for this section, we denote a rank vector as a vector with two elements, where the first element corresponds to element 1 of r and the second element corresponds to element 3 of r. The lowest rank is $(0, 0)$, followed by $(1, 0)$, the highest rank is $\top$, preceded by (n_1, n_3). Throughout the algorithm, whenever a vertex is contained in the set S_r, then its rank in the progress measure is at least r. The sets S_r are defined to contain vertices with rank *at least* r instead of *exactly* r such that for each vertex v and rank r' we only have to consider one set S_r in order to decide whether the rank of v can be increased to r'. Each vertex can only be assigned ranks for which all indices corresponding to priorities lower than its own priority are zero. For example, a vertex with priority 4 can only be assigned rank $(0, 0)$ or rank $\top$. Thus for such a vertex its rank can "jump" from $(0, 0)$ to $\top$ (because of one of its successors being added to $S_\top$), and then this information has to be propagated to its predecessors. The algorithm achieves this efficiently by adding, e.g., vertices that are added to $S_\top$ to all sets representing lower ranks as well. The sets S_r implicitly assign each vertex v a rank, namely the maximum rank r such that $v \in S_r$. We next describe the intuition for what it means when a vertex v is assigned a specific rank r.

- Intuitively, when a vertex v is assigned a rank $(i, 0)$ for $i \in N_1$, i.e., the set with highest rank it is contained in is $S_{(i,0)}$, then, in plays starting from v,

Algorithm SymbolicProgressMeasureParity(5):

```
Input  : parity game $\mathcal{P} = (\mathcal{G}, \alpha)$, with
         game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ and
         priority function $\alpha : V \to \{0, 1, 2, 3, 4\}$
Output : winning set of player $\mathcal{E}$
1  $S_{(0,0)} \leftarrow V$
2  $S_r \leftarrow \emptyset$ for $(0,0) < r \le (n_1, n_3)$ and $r = \top$
3  $r \leftarrow (1, 0)$
4  while true do
5      if $r = (i, j), i > 0$ then
6          $S_{(i,j)} \leftarrow S_{(i,j)} \cup \big(CPre_{\mathcal{O}}(S_{i-1,j}) \cap P_1\big)$
7          repeat
8              $S_{(i,j)} \leftarrow S_{(i,j)} \cup \big(CPre_{\mathcal{O}}(S_{i,j}) \setminus \bigcup_{i=2,3,4} P_i\big)$
9          until a fixed-point for $S_{(i,j)}$ is reached
10     else if $r = (0, j)$ then
11         $S_{(0,j)} \leftarrow S_{(0,j)} \cup \Big(CPre_{\mathcal{O}}(S_{n_1,j-1}) \cap P_1\Big) \cup \big(CPre_{\mathcal{O}}(S_{0,j-1}) \cap P_3\big)$
12         repeat
13             $S_{(0,j)} \leftarrow S_{(0,j)} \cup \big(CPre_{\mathcal{O}}(S_{0,j}) \setminus P_4\big)$
14         until a fixed-point for $S_{(0,j)}$ is reached
15     else if $r = \top$ then
16         $S_\top \leftarrow S_\top \cup \Big(CPre_{\mathcal{O}}(S_{n_1,n_3}) \cap P_1\Big) \cup \Big(CPre_{\mathcal{O}}(S_{0,n_3}) \cap P_3\Big)$
17         repeat
18             $S_\top \leftarrow S_\top \cup CPre_{\mathcal{O}}(S_\top)$
19         until a fixed-point for $S_\top$ is reached
20     $r' \leftarrow \mathrm{dec}(r)$
21     if $S_{r'} \supseteq S_r$ and $r < \top$ then
22         $r \leftarrow \mathrm{inc}(r)$
23     else if $S_{r'} \supseteq S_r$ and $r = \top$ then
24         break
25     else                              /* ensure $S_{r'} \supseteq S_r$ for all $r' < r$ */
26         repeat
27             $S_{r'} \leftarrow S_{r'} \cup S_r$
28             $r' \leftarrow \mathrm{dec}(r')$;
29         until $S_{r'} \supseteq S_r$
30         $r \leftarrow \mathrm{inc}(r')$        /* $S_r$ is modified set with smallest rank */
31 return $V \setminus S_\top$
```

player $\mathcal{O}$ can force a play from v to visit i vertices of priority 1 before a vertex of priority 2 or higher is reached; for $i > 0$ this implies that the priority of v is either 0 or 1.

- Let vertex v be assigned rank (i, j) for $i \in N_1$ and $j \in N_3$. The interpretation of the value of i is the same as in the case $(i, 0)$, where the value of i is taken modulo $n_1 + 1$. Each contribution of 1 to the value of j corresponds either (a) to the number of times a priority-3 vertex is visited before a priority-4 vertex is reached or (b) to priority-1 vertices being reached $n_1 + 1$ times before a vertex with priority at least 2 is reached. Note that case (b) can also happen for a vertex v with priority 2 or 3; for $\alpha(v) = 4$ the only set $S_{(i,j)}$ the vertex v can belong to is $S_{(0,0)}$, as the rank of a priority-4 vertex can only be $(0, 0)$ or $\top$.
- Recall that $\top$ are the vertices where player $\mathcal{E}$ has no winning strategy. There are three ways a vertex v can be ranked $\top$: (i) v has priority-1 and the best successor is ranked (n_1, n_3), (ii) v has priority-3 and the best successor is ranked at least $(0, n_3)$, and (iii) the best successor is ranked $\top$. The case (i) and (ii) correspond to the cases where player $\mathcal{E}$ has to visit at least $n_1 + 1$ or $n_3 + 1$ many vertices of the respective odd priority before reaching a higher even priority. Note that this means that player $\mathcal{O}$ can force plays to reach a cycle where the highest priority is odd. In case (iii) player $\mathcal{O}$ can force plays to visit at vertex from which she can reach a cycle with highest odd priority.

Symbolic algorithm. In our symbolic algorithm SymbolicProgressMeasureParity(5) we use the sets S_r to represent the numerical values of the progress measure, and, to utilize the power of symbolic operations, we compute all vertices whose rank can be increased to a certain value r in each iteration of the algorithm. The latter is in contrast to the explicit progress measure algorithm [Jur00], where vertices are considered one by one and the rank is increased to the maximal possible value.

We next describe Algorithm SymbolicProgressMeasureParity(5) and then give some intuition for its correctness and the number of its symbolic operations. Recall that the sets S_r define a ranking function $\rho_{\{S_r\}_r}(v) = \max\{r \in M^{\infty}_{\mathcal{G}} \mid v \in S_r\}$ that assigns a rank to each vertex and that by the definition of inc (Section 5.2.6.2) a vertex v with priority $\alpha(v)$ is only assigned ranks r with $r = \langle r \rangle_{\alpha(v)}$.

Initialization. To find the least simultaneous fixed point of the lift-operators, all ranks are initialized with the all zero vector, i.e., all vertices are added to $S_{(0,0)}$, while all other sets S_r are empty. The variable r is initialized to $(1, 0)$.

The value of r. In each iteration of the while-loop the set S_r for the rank that is stored in the variable r at the beginning of the iteration is updated (more details below). By the definition of the sets $S_{r'}$ we need to maintain $S_{r'} \supseteq S_r$ for $r' < r$, i.e., every vertex that is newly added to S_r but not yet contained in $S_{r'}$ is added to $S_{r'}$ (line 27). For the vertices newly added to S_r we say that their rank is increased. When the rank of a vertex is increased, this might influence the value of $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v)$ for its neighbors v. Since we want to obtain a fixed point of $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v)$ for all $v \in V$, we have to reconsider the neighbors of a vertex whenever the rank of the vertex is increased. This is achieved by updating the variable r to the lowest r' for which a new vertex is added to $S_{r'}$ in this iteration (lines 26–30). If $r = r'$, then for $r < \top$ the value of r is increased to the next highest rank in the ordering (line 22) and for $r = \top$ the algorithm terminates (line 24).

Update of set S_r. To reach a simultaneous fixed point of the lift-operators, the rank of a vertex v has to be increased to $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v)$ whenever the value of $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v)$ is strictly higher than $\rho_{\{S_r\}_r}(v)$ for the current ranking function $\rho_{\{S_r\}_r}$. Recall that this is the case when the value of $\mathrm{best}(\rho, v)$ is increased, which implies that the rank assigned to at least one successor of v is increased. For the update of the set S_r in an iteration of the while-loop (where r is the value of the variable at the beginning of the while-loop), the algorithm adds vertices with (i) $\rho_{\{S_r\}_r}(v) < r$, (ii) $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$, and (iii) $r = \langle r \rangle_{\alpha(v)}$ to S_r. Note that the algorithm maintains the invariant that vertices with $\rho_{\{S_r\}_r} \geq r$ are already contained in S_r. We distinguish between $r = (i, j)$ with $i > 0$, $r = (0, j)$, and $r = \top$.

(1) For $r = (i, j)$ with $i > 0$ recall that only vertices v with priority 0 or 1 can be assigned rank r.

Case (a): Assume first that $\alpha(v) = 1$. Then $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ if (i) $v \in V_{\mathcal{E}}$ and all successors w of v have $\rho_{\{S_r\}_r}(w) \geq (i-1, j)$ or (ii) $v \in V_{\mathcal{O}}$ and one successor w of v has $\rho_{\{S_r\}_r}(w) \geq (i-1, j)$. That is, $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ only if $v \in \mathit{CPre}_{\mathcal{O}}(S_{(i-1,j)})$. In this case the vertex v is added to S_r in line 6.

Case (b): Assume now that $\alpha(v) = 0$. Then $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ if (i) $v \in V_{\mathcal{E}}$ and all successors w of v have $\rho_{\{S_r\}_r}(w) \geq (i, j)$ or (ii) $v \in V_{\mathcal{O}}$ and one successor w of v has $\rho_{\{S_r\}_r}(w) \geq (i, j)$. That is, $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ only if $v \in \mathit{CPre}_{\mathcal{O}}(S_{(i,j)})$. In this case the vertex v is added to S_r in some iteration of the repeat-until loop in line 8.

(2) The difference for $r = (0, j)$ is first that also vertices with priorities 2 or 3 are candidates for the assignment of rank r and second that for a vertex v with odd priority we now have the following possibilities: either (i) $\alpha(v) = 1$ and we consider neighbors w with $\rho_{\{S_r\}_r}(w) \geq (n_1, j-1)$ or (ii) $\alpha(v) = 3$ and we consider neighbors w with $\rho_{\{S_r\}_r}(w) \geq (0, j-1)$.

(3) The case $r = \top$ corresponds to the case $r = (0, j)$ with $j = n_3 + 1$ with the difference that also vertices with priority 4 can be included in $S_\top$ in the case $\rho_{\{S_r\}_r}(w) = \top$.

Sketch of bound on number of symbolic operations. Observe that each rank r is considered in at least one iteration of the while-loop but is only reconsidered in a later iteration if at least one vertex was added to the set S_r since the last time r was considered. This implies by $|S_r| \leq n$ that Algorithm SymbolicProgressMeasureParity(5) can be implemented with $O(n \cdot |M^{\infty}_{\mathcal{G}}|) = O(n^3)$ symbolic operations.

Sketch of correctness. Let $\{S_r\}_r$ be the sets in the algorithm at termination. The algorithm returns the set of vertices that are not contained in $S_\top$, i.e., the vertices to which $\rho_{\{S_r\}_r}$ assigns a rank $< \top$. If $\rho_{\{S_r\}_r}$ is equal to the progress measure of the parity game, then by Lemma 5.2.4 the returned set $V \setminus S_\top$ is equal to the winning set of player $\mathcal{E}$. Let $\tilde{\rho}$ denote the progress measure. It remains to show that $\rho_{\{S_r\}_r}(v) = \tilde{\rho}(v)$ for all $v \in V$ when the algorithm terminates. To this end we show (1) that the algorithm only adds a vertex to a set S_r when the progress measure $\tilde{\rho}$ is at least r, i.e., throughout the algorithm the ranking function $\rho_{\{S_r\}_r}$ is a lower bound on $\tilde{\rho}$ and (2) by the update of the variable r we have before and after each iteration of the while-loop for all vertices v that either $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$ or $\rho_{\{S_r\}_r}(v) \geq r$ and in the final iteration with $r = \top$ we have $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$ for all $v \in V$. This implies that when the algorithm terminates the ranking function $\rho_{\{S_r\}_r}$ is a simultaneous fixed point of the lift-operators. Together these two properties imply that the algorithm computes the progress measure of the parity game.

5.4.2 General Parity Games

In this section we generalize the algorithm for parity games with 5 priorities to parity games with c priorities and provide proofs for the correctness and the number of symbolic operations of the algorithm. We first present the variant that uses an exponential number of sets and then, in Section 5.4.3, show how to reduce the number of sets to linear.

Key differences and challenges. We mention the key differences of our algorithm and the progress-measure algorithm, and the associated challenges.

- First, in our symbolic SymbolicParityDominion algorithm we represent the numerical values of the progress measure by sets S_r storing all vertices with rank at least r.
- Second, to exploit the power of symbolic operations, in each iteration of the algorithm we compute all vertices whose rank can be increased to a certain value r. This is in sharp contrast to the explicit progress-measure algorithm [Jur00], where vertices are considered one by one and the rank is increased to the maximal possible value.

The above gives a set-based symbolic algorithm, but since now we deal with sets of vertices, as compared to individual vertices, the correctness needs to be established. The non-trivial aspect of the proof is to identify appropriate *invariants on sets* (which we call *symbolic invariants*, see Invariant 5.4.1) and use them to establish the correctness.

The co-domain M_h^∞. Additionally to the generalization to an arbitrary number of priorities, we formulate our algorithm such that it can not only compute the winning sets of the players but also $\mathcal{E}$-dominions of size at most $h + 1$. (For $\mathcal{O}$-dominions add one to each priority and exchange the roles of the two players.) The only change needed for this is to use the co-domain M_h^∞, instead of $M_\mathcal{G}^\infty$, for the inc and dec operations. The co-domain M_h^∞ contains all ranks of $M_\mathcal{G}^\infty$ whose entries sum up to at most h (see Section 5.2.6.3).

Algorithm SymbolicParityDominion: Symbolic progress measure algorithm for parity games

```
Input  : parity game $\mathcal{P} = (\mathcal{G}, \alpha)$, with
         game graph $\mathcal{G} = ((V, E), (V_\mathcal{E}, V_\mathcal{O}))$ and
         priority function $\alpha : V \to [c]$, and
         parameter $h \in [0, n] \cap \mathbb{N}$
Output : an $\mathcal{E}$-dominion that contains all $\mathcal{E}$-dominions of size $\leq h + 1$ or possibly
         the empty set if no such $\mathcal{E}$-dominion exists
1  $S_{\bar{0}} \leftarrow V$; $S_r \leftarrow \emptyset$ for $r \in M_h^\infty \setminus \{\bar{0}\}$
2  $r \leftarrow \mathrm{inc}(\bar{0})$
3  while true do
4      if $r \neq \top$ then
5          Let $\ell$ be maximal such that $r = \langle r \rangle_\ell$
6          $S_r \leftarrow S_r \cup \bigcup_{1 \leq k \leq (\ell+1)/2} \left( CPre_\mathcal{O}(S_{\mathrm{dec}_{2k-1}(r)}) \cap P_{2k-1} \right)$
7          repeat
8              $S_r \leftarrow S_r \cup \left( CPre_\mathcal{O}(S_r) \setminus \bigcup_{\ell < k < c} P_k \right)$
9          until a fixed-point for $S_r$ is reached
10     else if $r = \top$ then
11         $S_\top \leftarrow S_\top \cup \bigcup_{1 \leq k \leq \lfloor c/2 \rfloor} \left( CPre_\mathcal{O}(S_{\mathrm{dec}_{2k-1}(\top)}) \cap P_{2k-1} \right)$
12         repeat
13             $S_\top \leftarrow S_\top \cup \left( CPre_\mathcal{O}(S_\top) \right)$
14         until a fixed-point for $S_\top$ is reached
15     $r' \leftarrow \mathrm{dec}(r)$
16     if $S_{r'} \supseteq S_r$ and $r < \top$ then
17         $r \leftarrow \mathrm{inc}(r)$
18     else if $S_{r'} \supseteq S_r$ and $r = \top$ then
19         break
20     else
21         repeat
22             $S_{r'} \leftarrow S_{r'} \cup S_r$
23             $r' \leftarrow \mathrm{dec}(r')$;
24         until $S_{r'} \supseteq S_r$
25         $r \leftarrow \mathrm{inc}(r')$
26 return $V \setminus S_\top$
```

The sets S_r and the ranking function $\rho_{\{S_r\}_r}$. The algorithm implicitly maintains a rank for each vertex. A vertex is contained in a set S_r only if its maintained rank is at least r. Each set S_r is monotonically increasing throughout the algorithm. Furthermore, the algorithm maintains *anti-monotonicity* among the sets, i.e., we have $S_{r'} \supseteq S_r$ for all r and all $r' < r$ before and after each iteration. The rank of a vertex v is the highest r such that $v \in S_r$. In other words, the family of sets $\{S_r\}_r$ defines the ranking function $\rho_{\{S_r\}_r}(v) = \max\{r \in M_h^\infty \mid v \in S_r\}$.

Structure of the algorithm. The overall structure of the algorithm is the same as for parity games with 5 priorities: The set $S_{\bar{0}}$ is initialized with the set of all vertices V, while all other set S_r for $r > \bar{0}$ are initially empty, i.e., the ranks of all vertices are initialized with the zero vector. The variable r is initially set to the second lowest rank $\mathrm{inc}(\bar{0})$ that is one at index 1 and zero otherwise. In the while-loop the set S_r is updated for the value of r at the beginning of the iteration (see below). After the update of S_r, it is checked whether the set corresponding to the next lowest rank already contains the vertices newly added to S_r, i.e., whether the anti-monotonicity is preserved. If the anti-monotonicity is preserved despite the update of S_r, then for $r < \top$ the value of r is increased to the next highest rank (line 17) and for $r = \top$ the algorithm terminates (line 19). Otherwise the vertices newly added to S_r are also added to all sets with $r' < r$ that do not already contain them; the variable r is then updated to the lowest r' for which a new vertex is added to $S_{r'}$ in this iteration (lines 21–25).

Update of set S_r. Consider a fixed iteration of the while-loop and let S_r be the set that is updated in this iteration in lines 4–14. Let $\rho_{\{S_r\}_r}$ be denoted by ρ for short. In this update of the set S_r we want to add to S_r all vertices v with $\rho(v) < r$ and $\mathtt{Lift}(\rho, v)(v) \geq r$ under the condition that the priority of v allows v to be assigned the rank r, i.e., $r = \langle r \rangle_{\alpha(v)}$. Note that by the anti-monotonicity property the set S_r already contains all vertices with $\rho(v) \geq r$.

(1) We fist consider the case $r < \top$ (lines 4–9). Let ℓ be maximal such that $r = \langle r \rangle_\ell$, i.e., the first ℓ entries with indices 0 to $\ell - 1$ of r are 0 and the entry with index ℓ is larger than 0. Note that ℓ is odd. We have that only the $\mathtt{Lift}(., v)$-operators with $\alpha(v) \leq \ell$ can increase the rank of a vertex to r as all the others would set the element with index ℓ to 0.

Recall that $\mathtt{Lift}(\rho, v)(v) = \mathrm{inc}_{\alpha(v)}(\mathrm{best}(\rho, v))$, where best considers the vertices with edges from v and is implemented by the $\mathit{CPre}_\mathcal{O}$ operator. The function $\mathrm{inc}_{\alpha(v)}(x)$ for $x < \top$ behaves differently for odd and even $\alpha(v)$ (see Section 5.2.6.2): If $\alpha(v)$ is odd, then $\mathrm{inc}_{\alpha(v)}(x)$ is the smallest rank y in M_h^∞ such that $y >_{\alpha(v)} x$, i.e., y is larger than x w.r.t. indices $\geq \alpha(v)$. If $\alpha(v)$ is even, then $\mathrm{inc}_{\alpha(v)}(x)$ is equal to x with the indices lower than $\alpha(v)$ set to 0.

- First, consider a $\mathtt{Lift}(\rho, v)$ operation with odd $\alpha(v) \leq \ell$, i.e., let $\alpha(v) = 2k - 1$ for some $1 \leq k \leq (\ell + 1)/2$. Then $\mathtt{Lift}(\rho, v)(v) \geq r$ only if (a) $v \in$

$V_{\mathcal{E}}$ and all successors w have $\rho(w) \geq \mathrm{dec}_{2k-1}(r)$, or (b) $v \in V_{\mathcal{O}}$ and one successor w has $\rho(w) \geq \mathrm{dec}_{2k-1}(r)$. That is, $\mathtt{Lift}(\rho, v)(v) \geq r$ only if $v \in CPre_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)})$. Vice versa, we have that if $v \in CPre_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)})$ then by $\rho = \rho_{\{S_r\}_r}$ also $\mathtt{Lift}(\rho, v)(v) \geq r$. This observation is implemented in SymbolicParityDominion in line 6, where such vertices v are added to S_r.

- Now, consider a $\mathtt{Lift}(\rho, v)$ operation with even $\alpha(v) \leq \ell$, i.e., let $\alpha(v) = 2k$ for some $1 \leq k \leq \ell/2$. Then $\mathtt{Lift}(\rho, v)(v) \geq r$ only if (a) $v \in V_{\mathcal{E}}$ and all successors w have $\rho(w) \geq r$, or (b) $v \in V_{\mathcal{O}}$ and one successor w has $\rho(w) \geq r$. That is, $\mathtt{Lift}(\rho, v)(v) \geq r$ only if $v \in CPre_{\mathcal{O}}(S_r)$. Vice versa, we have that if $v \in CPre_{\mathcal{O}}(S_r)$ then $\mathtt{Lift}(\rho, v)(v) \geq r$. In SymbolicParityDominion these vertices are added iteratively in line 8 until a fixed point is reached. Notice that in line 8 the algorithm also adds vertices v with odd priority to S_r, but due do the above argument we have $\mathtt{Lift}(\rho, v)(v) > r$ and thus they can be included in S_r.

(2) The case $r = \top$ (lines 10–14) works similarly except that (a) every vertex is a possible candidate for being assigned the rank $\top$, independent of its priority (line 11) and (b) whenever x is equal to $\top$, $\mathrm{inc}_{\alpha(v)}(x)$ assigns the rank $\top$ independently of $\alpha(v)$ (line 13).

Sketch of bound on number of symbolic operations. As for parity games with 5 priorities, the number of symbolic operations per set S_r is of the same order as the number of times a vertex is added to the set.[2] Thus the algorithm can be implemented with $O(n \cdot |M_h^\infty|)$ symbolic operations. For the co-domain $M_{\mathcal{G}}^\infty$ the bound $O(n \cdot |M_{\mathcal{G}}^\infty|)$ is analogous.

Outline correctness proof. In the following proof we show that when Algorithm SymbolicParityDominion terminates, the ranking function $\rho_{\{S_r\}_r}$ is equal to the progress measure for the given parity game and the co-domain M_h^∞. The same proof applies for the co-domain $M_{\mathcal{G}}^\infty$. The algorithm returns the set of vertices that are assigned a rank $< \top$ when the algorithm terminates. By Lemma 5.2.7 this set is an $\mathcal{E}$-dominion that contains all $\mathcal{E}$-dominions of size at most $h+1$ when the co-domain M_h^∞ is used, and by Lemma 5.2.4 this set is equal to the winning set of player $\mathcal{E}$ when the co-domain $M_{\mathcal{G}}^\infty$ is used. Thus it remains to show that $\rho_{\{S_r\}_r}$ equals the progress measure for the given co-domain when the algorithm terminates. We first show that maintaining the following invariants over all iteration of the algorithm is sufficient for this and then prove that the invariants are maintained. All proofs are described for the co-domain M_h^∞.

[2]Notice that to initialize Algorithm SymbolicParityDominion, more precisely to compute the set $M_{\mathcal{G}}^\infty$, we need to determine the sizes of the sets P_i. Such a cardinality operation $|S|$ returning the number of elements of a set S is provided by standard symbolic implementations, e.g., see [Cha+13, Section 3.2]. Also we need only $O(c)$ cardinality operations.

Invariant 5.4.1 (Symbolic invariants). *In Algorithm SymbolicParityDominion the following three invariants hold. Every rank is from the co-domain M_h^∞ and the* $\mathtt{Lift}(.,v)$*-operators are defined w.r.t. the co-domain. Let $\tilde{\rho}$ be the progress measure of the given parity game and let $\rho_{\{S_r\}_r} = \max\{r \in M_h^\infty \mid v \in S_r\}$ be the ranking function with respect to the sets S_r that are maintained by the algorithm.*

(1) *Before and after each iteration of the while-loop we have that if a vertex v is in a set S_{r_1} then it is also in S_{r_2} for all $r_2 < r_1$ (anti-monotonicity).*

(2) *Throughout Algorithm SymbolicParityDominion we have $\tilde{\rho}(v) \geq \rho_{\{S_r\}_r}(v)$ for all $v \in V$.*

(3) *Before and after each iteration of the while-loop we have for the rank stored in r and all vertices v either* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ *or* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$. *At line 15 of the algorithm we additionally have $v \in S_r$ for all vertices v with* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r$ *(closure property).*

Informal description of invariants. Invariant 5.4.1(1) ensures that the definition of the sets S_r and the ranking function $\rho_{\{S_r\}_r}$ is sound; Invariant 5.4.1(2) guarantees that $\rho_{\{S_r\}_r}$ is a lower bound on $\tilde{\rho}$ throughout the algorithm; and Invariant 5.4.1(3) shows that when the algorithm terminates, a fixed point of the $\rho_{\{S_r\}_r}$ function with respect to the $\mathtt{Lift}(.,v)$-operators is reached. Together these three properties guarantee that when the algorithm terminates, the function $\rho_{\{S_r\}_r}$ corresponds to the progress measure, i.e., to the least simultaneous fixed point of the $\mathtt{Lift}(.,v)$-operators. We start with proving this claim and then prove each of the invariants.

Lemma 5.4.2. *Assuming that Invariant 5.4.1 holds, the ranking function $\rho_{\{S_r\}_r}$ induced by the family of sets $\{S_r\}_r$ at termination of Algorithm SymbolicParityDominion is equal to the progress measure for the given parity game and the co-domain M_h^∞.*

Proof. Recall that the progress measure is the least simultaneous fixed point of all $\mathtt{Lift}(.,v)$-operators for the given parity game (where inc, dec, and the ordering of ranks are w.r.t. the given co-domain) and let the progress measure be denoted by $\tilde{\rho}$. Let $\{S_r\}_r$ be the sets in the algorithm at termination. For all $v \in V$ the ranking function $\rho_{\{S_r\}_r}(v)$ is defined as $\max\{r \in M_h^\infty \mid v \in S_r\}$. By Invariant 5.4.1(2) we have $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ for all $v \in V$.

When the algorithm terminates, with $r = \top$, we have by Invariant 5.4.1(3) $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$ for each vertex v and thus $\rho_{\{S_r\}_r}$ is a simultaneous fixed point of the $\mathtt{Lift}(.,v)$-operators. Now, as $\tilde{\rho}$ is the least simultaneous fixed point of all $\mathtt{Lift}(.,v)$-operators, we obtain $\rho_{\{S_r\}_r}(v) \geq \tilde{\rho}(v)$ for all $v \in V$. Hence we have $\rho_{\{S_r\}_r}(v) = \tilde{\rho}(v)$ for all $v \in V$. □

We prove the three parts of Invariant 5.4.1 with the following three lemmata. Notice that Invariant 5.4.1(1) is required to prove Invariant 5.4.1(3).

Lemma 5.4.3. *Before and after each iteration of the while-loop in Algorithm SymbolicParityDominion we have $S_{r_1} \supseteq S_{r_2}$ for all $r_1 \leq r_2$ with $r_1, r_2 \in M_h^\infty$, i.e., Invariant 5.4.1((1)) holds.*

Proof. The proof is by induction over the iterations of the while-loop. The claim is satisfied when we first enter the while-loop and only $S_{\bar{0}}$ is non-empty. It remains to show that when the claim is valid at the beginning of a iteration then the claim also hold afterwards. By the induction hypothesis, the sets $S_{r'}$ for $r' < r$ are monotonically decreasing. Thus it is sufficient to find the lowest rank r^* such that for all $r^* \leq r' < r$ we have $S_r \not\subseteq S_{r'}$ and add the vertices newly added to S_r to the sets $S_{r'}$ with $r^* \leq r' < r$, which is done in lines 16–25 of the while-loop. □

Lemma 5.4.4. *Let $\tilde{\rho}$ be the progress measure of the given parity game and let $\rho_{\{S_r\}_r} = \max\{r \in M_h^\infty \mid v \in S_r\}$ be the ranking function with respect to the family of sets $\{S_r\}_r$ that is maintained by the algorithm. Throughout Algorithm SymbolicParityDominion we have $\tilde{\rho}(v) \geq \rho_{\{S_r\}_r}(v)$ for all $v \in V$, i.e., Invariant 5.4.1((2)) holds.*

Proof. We show the lemma by induction over the iterations of the while-loop. Before the first iteration of the while-loop only $S_{\bar{0}}$ is non-empty, thus the claim holds by $\tilde{\rho} \geq \bar{0}$.

Assume we have $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ for all $v \in V$ before an iteration of the while-loop. We show that $\rho_{\{S_r\}_r}(v) \leq \tilde{\rho}(v)$ also holds during and after the iteration of the while-loop. As the update of $S_{r'}$ in line 22 does not change $\rho_{\{S_r\}_r}$, we only have to show that the invariant is maintained by the update of S_r in lines 4–14. Further $\rho_{\{S_r\}_r}(v)$ only changes for vertices newly added to S_r, thus we only have to take these vertices into account.

Assume $r < \top$, the argument for $r = \top$ is analogous. Recall that ℓ is the maximal index such that $r = \langle r \rangle_\ell$.[3] The algorithm adds vertices to S_r in (1) line 6 and (2) line 8. In case (1) we add the vertices $\bigcup_{1 \leq k \leq (\ell+1)/2} (CPre_{\mathcal{O}}(S_{\mathrm{dec}_{2k-1}(r)}) \cap P_{2k-1})$ to S_r. Let $v \in CPre_{\mathcal{O}}\,(S_{\mathrm{dec}_{2k-1}(r)}) \cap P_{2k-1}$ for some $1 \leq k \leq (\ell + 1)/2$.

- If $v \in V_{\mathcal{E}} \cap P_{2k-1}$, then all successors w of v are in $S_{\mathrm{dec}_{2k-1}(r)}$ and thus, by the induction hypothesis, have $\tilde{\rho}(w) \geq \mathrm{dec}_{2k-1}(r)$. Now as $v \in P_{2k-1}$, it has rank $\tilde{\rho}(v)$ at least $\mathrm{inc}_{2k-1}(\mathrm{dec}_{2k-1}(r)) = r$.

- If $v \in V_{\mathcal{O}} \cap P_{2k-1}$, at least one successors w of v is in $S_{\mathrm{dec}_{2k-1}(r)}$ and thus, by the induction hypothesis, has $\tilde{\rho}(w) \geq \mathrm{dec}_{2k-1}(r)$. Now as $v \in P_{2k-1}$, it has rank $\tilde{\rho}(v)$ at least $\mathrm{inc}_{2k-1}(\mathrm{dec}_{2k-1}(r)) = r$.

For case (2) consider a vertex $v \in CPre_{\mathcal{O}}(S_r) \setminus \bigcup_{\ell < k \leq d} P_k$ added in line 8.

- If $v \in V_{\mathcal{E}}$, all successors w of v are in S_r and thus, by the induction hypothesis, have $\tilde{\rho}(w) \geq r$. Since the priority of v is $\leq \ell$, we have $\tilde{\rho}(v) \geq \langle r \rangle_\ell = r$.

[3]For the case $r = \top$, let ℓ be the highest odd priority.

- If $v \in V_{\circledcirc}$, at least one successors w of v is in S_r and thus, by the induction hypothesis, has $\tilde{\rho}(w) \geq r$. Since the priority of v is $\leq \ell$, we have $\tilde{\rho}(v) \geq \langle r \rangle_{\ell} = r$. □

Lemma 5.4.5. *Before and after each iteration of the while-loop, we have for the rank stored in r and all vertices v either* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) \geq r$ *or* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = \rho_{\{S_r\}_r}(v)$. *At line 15 of the algorithm we additionally have $v \in S_r$ for all vertices v for which the value of* $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v)$ *is equal to r. Thus Invariant 5.4.1((3)) holds.*

Proof. We show the claim by induction over the iterations of the while-loop. Before we first enter the loop, we have $r = \text{inc}(\bar{0})$ and $S_{\bar{0}} = V$ and thus the claim is satisfied. For the inductive step, let r^{old} be the value of r and ρ^{old} the ranking function $\rho_{\{S_r\}_r}$ before a fixed iteration of the while-loop and assume we have for all $v \in V$ either $\mathtt{Lift}(\rho^{\text{old}}, v)(v) \geq r^{\text{old}}$ or $\mathtt{Lift}(\rho^{\text{old}}, v)(v) = \rho^{\text{old}}(v)$ before the iteration of the while-loop. Let r^{new} be the value of r and ρ^{new} the ranking function $\rho_{\{S_r\}_r}$ after the iteration. We have three cases for the value of r^{new}: (1) $r^{\text{new}} = \text{inc}(r^{\text{old}})$ (line 16), (2) $r^{\text{new}} = r^{\text{old}} = \top$ (line 18), or (3) $r^{\text{new}} < r^{\text{old}}$, i.e., the rank is decreased in lines 21–25 to maintain anti-monotonicity.

We show in Claim 5.4.6 that, in all three cases, if a set $S_{r'}$, for some $r' < r^{\text{old}}$, is not changed in the considered iteration of the while-loop then for all $v \in V$ with $\mathtt{Lift}(\rho^{\text{new}}, v)(v) \leq r'$ we have that $\mathtt{Lift}(\rho^{\text{new}}, v)(v) = \rho^{\text{new}}(v)$.

Given Claim 5.4.6, we prove the first part of the invariant as follows. In the case (1) the lowest (and only) rank for which the set is updated is r^{old}, thus it remains to show $\mathtt{Lift}(\rho^{\text{new}}, v)(v) = \rho^{\text{new}}(v)$ for vertices with $\mathtt{Lift}(\rho^{\text{new}}, v)(v) = r^{\text{old}}$, which is done by showing the second part of the invariant, namely that $v \in S_{r^{\text{old}}}$ for all vertices v with $\mathtt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\text{old}}$ after the update of the set $S_{r^{\text{old}}}$ in lines 4–14; for case (1) we have $\rho^{\text{new}} = \rho_{\{S_r\}_r}$ at this point in the algorithm.

In the cases (2) and (3) we have that the lowest rank for which the set is updated in the iteration is equal to r^{new}, thus Claim 5.4.6 implies that the invariant $\mathtt{Lift}(\rho^{\text{new}}, v)(v) \geq r^{\text{new}}$ or $\mathtt{Lift}(\rho^{\text{new}}, v)(v) = \rho^{\text{new}}(v)$ holds for all $v \in V$ after the while-loop.

Claim 5.4.6. *Let $r^* \leq r^{old}$ be a rank with the guarantee that no set corresponding to a lower rank than r^* is changed in this iteration of the while-loop. Then we have for all $v \in V$ either* $\mathtt{Lift}(\rho^{new}, v)(v) \geq r^*$ *or* $\mathtt{Lift}(\rho^{new}, v)(v) = \rho^{new}(v)$ *after the iteration of the while-loop.*

To prove the claim, note that since each set S_r is monotonically non-decreasing over the algorithm, we have $\rho^{\text{new}}(v) \geq \rho^{\text{old}}(v)$ and $\mathtt{Lift}(\rho^{\text{new}}, v)(v) \geq \rho^{\text{new}}(v)$. Assume by contradiction that there is a vertex v with $\mathtt{Lift}(\rho^{\text{new}}, v)(v) > \rho^{\text{new}}(v)$ and $\mathtt{Lift}(\rho^{\text{new}}, v)(v) < r^*$. The latter implies $\text{best}(\rho^{\text{new}}, v) < r^*$. By the induction hypothesis for $r^* \leq r^{\text{old}}$ we have $\mathtt{Lift}(\rho^{\text{old}}, v)(v) = \rho^{\text{old}}(v)$. By $\rho^{\text{new}}(v) \geq \rho^{\text{old}}(v)$ and the definition of the lift-operator this implies $\text{best}(\rho^{\text{new}}, v) > \text{best}(\rho^{\text{old}}, v)$, i.e., the rank assigned to at least one vertex w with $(v, w) \in E$ is increased. By $\text{best}(\rho^{\text{new}}, v) < r^*$

this implies that a set $S_{r'}$ with $r' < r^*$ is changed in this iteration, a contradiction to the definition of r^*. This concludes the proof of the claim.

It remains to show that $v \in S_{r^{\text{old}}}$ for all vertices v with $\texttt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\text{old}}$ after the update of the set $S_{r^{\text{old}}}$ in lines 4–14. Towards a contradiction assume that there is a $v \notin S_{r^{\text{old}}}$ such that $\texttt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\text{old}}$. Assume $v \in V_{\mathcal{E}}$, the argument for $v \in V_{\mathcal{O}}$ is analogous. Let ℓ be maximal such that $r^{\text{old}} = \left\langle r^{\text{old}} \right\rangle_{\ell}$ for $r^{\text{old}} < \top$ and let ℓ be the highest odd priority in the parity game for $r^{\text{old}} = \top$. Notice that $\alpha(v)$ can be at most ℓ for $\texttt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\text{old}}$ to hold. We now distinguish two cases depending on whether $\alpha(v)$ is odd or even.

- If $\alpha(v)$ is odd, i.e., $\alpha(v) = 2k - 1$ for some $k \leq (\ell + 1)/2$, then we have that all successors w of v have $\rho_{\{S_r\}_r}(w) \geq \text{dec}_{2k-1}(r^{\text{old}})$ and thus, by Lemma 5.4.3, $w \in S_{\text{dec}_{2k-1}(r^{\text{old}})}$. But then v would have being included in $S_{r^{\text{old}}}$ in line 6, a contradiction.

- If $\alpha(v)$ is even, i.e., $\alpha(v) = 2k$ for some $k \leq \ell/2$, then we have that all successors w of v have $\rho_{\{S_r\}_r}(w) \geq r^{\text{old}}$. Then by the definition of $\rho_{\{S_r\}_r}$ it must be that $w \in S_{r'}$ for some $r' \geq r^{\text{old}}$ and by Lemma 5.4.3 it must be that $w \in S_{r^{\text{old}}}$. But then v would have being included in $S_{r^{\text{old}}}$ in line 8, a contradiction.

Thus, after the update of the set $S_{r^{\text{old}}}$ we have that $v \in S_{r^{\text{old}}}$ for all vertices v with $\texttt{Lift}(\rho_{\{S_r\}_r}, v)(v) = r^{\text{old}}$. Together we the above observations this proves the lemma. □

Combining Lemma 5.4.2 with Lemmata 5.4.3–5.4.5 that prove Invariant 5.4.1 concludes the correctness proof.

Lemma 5.4.7 (Correctness). *Algorithm SymbolicParityDominion computes the progress measure for a given parity game (with n vertices) and a given set of possible ranks M_h^∞ (for some integer $h \in [1, n-1]$).*

We address next the number of symbolic operations of Algorithm SymbolicParityDominion. The main idea is that a set S_r is only reconsidered after at least one new vertex is added to S_r.

Lemma 5.4.8. *For parity games with n vertices and c priorities Algorithm SymbolicParityDominion takes $O(n \cdot c \cdot |M_h^\infty|)$ many symbolic operations using $O(|M_h^\infty|)$ many sets, where h is some integer in $[1, n-1]$.*

Proof. In the algorithm we use one set S_r for each $r \in M_h^\infty$ and thus $|M_h^\infty|$ many sets. We first consider the number of symbolic operations needed to compute the sets S_r in lines 4–14, and then consider the number of symbolic operations to compute the new value of r in lines 15–25.

1) Whenever we consider a set S_r we first initialize the set with $O(c)$ many symbolic operations (lines 6 & 11). After that we do a fixed-point computation

that needs $O(k)$ symbolic operation, where k is the number of added vertices. Now fix a set S_r and consider all the fixed-point computation for S_r over the whole algorithm. As only $O(n)$ many vertices can be added to S_r, all these fixed-points can be computed in $O(n+\#r)$ symbolic operation, where $\#r$ is the number of times the set S_r is considered (the algorithm needs a constant number of symbolic operations to realize that a fixed-point was already reached). Each set S_r is considered at least once and only reconsidered when some new vertices are added to the set, i.e., it is considered at most n times. Thus for each set S_r we have $O(c \cdot n)$ many operations, which gives a total number of operations of $O(n \cdot c \cdot |M_h^\infty|)$.

2) Now consider the computation of the new value of r in lines 15–25. Lines 15–19 take a constant number of operations. It remains to count the iterations of the repeat-until loop in lines 20–25, which we bound by the number of iterations of the while-loop as follows. Whenever a set $S_{r'}$ is considered as the left side argument in line 24, then the new value for r is less or equal to $\mathrm{inc}(r')$ and thus there will be another iteration of the while-loop considering $\mathrm{inc}(r')$. As there are only $O(n \cdot |M_h^\infty|)$ many iterations of the while-loop over the whole algorithm, there are only $O(n \cdot |M_h^\infty|)$ many iterations of the repeat-until loop in total. In each iteration a constant number of operations is performed.

By (1) and (2) we have that Algorithm SymbolicParityDominion takes $O(n \cdot c \cdot \left|M_h^\infty\right|)$ many symbolic operations. □

If we want to compute dominions of size $h+1$, by Lemma 5.2.7, we only have to consider ranks in M_h^∞. To obtain our final theorem, we apply the following bound[4] on the size of M_h^∞:

$$|M_h^\infty| \le \binom{h + \lfloor c/2 \rfloor}{h} + 1$$

Key Lemma 5.4.9. *For a given parity game with n vertices and c priorities and an integer $h \in [1, n-1]$, Algorithm SymbolicParityDominion computes a player-$\mathcal{E}$ dominion that contains all $\mathcal{E}$-dominions with at most $h+1$ vertices with $O\left(c \cdot n \cdot \binom{h+\lfloor c/2 \rfloor}{h}\right)$ symbolic one-step and set operations.*

To solve parity games directly with Algorithm SymbolicParityDominion, we use the co-domain $M_{\mathcal{G}}^\infty$ instead of M_h^∞. Recall from Section 5.2.6.2 that we have $|M_{\mathcal{G}}^\infty| \in O\big(\big(\frac{n}{\lfloor c/2 \rfloor}\big)^{\lfloor c/2 \rfloor}\big)$ [Jur00].

[4] To bound $|M_h|$ we have to count the sequences of non-negative integers $x_1, x_2, \dots, x_{\lfloor c/2 \rfloor}$ such that $\sum_{i=1}^{\lfloor c/2 \rfloor} x_i \le h$. Such a sequence is equivalent to a weak composition of an integer h into exactly $\lfloor c/2 \rfloor + 1$ parts which gives the $\binom{h+\lfloor c/2 \rfloor}{h}$ bound.

Theorem 5.4.10. *Let $\xi(n,c) = \left(\frac{n}{\lfloor c/2 \rfloor}\right)^{\lfloor c/2 \rfloor}$. Algorithm SymbolicParityDominion correctly computes the winning sets of parity games and requires $O\big(c \cdot n \cdot \xi(n,c)\big)$ symbolic one-step and set operations and space for $O\big(\xi(n,c)\big)$ many sets.*

5.4.3 A Linear Space Algorithm

Algorithm SymbolicParityDominion requires $|M^{\infty}_{\mathcal{G}}|$ many sets S_r, which is drastically beyond the space requirement of the progress measure algorithm for explicitly represented graphs. The aim of this section is to reduce the space requirement to $O(n)$ many sets, in a way that still allows to restore the sets S_r efficiently. For the sake of readability, we assume for this section that c is even.

Main idea. The main idea to reduce the space requirement for the symbolic progress measure algorithm is as follows.

- Instead of storing sets S_r corresponding to a specific rank, we encode the value of each coordinate of the rank r separately, i.e., we use sets C^i_x containing all vertices with a rank where the i-th coordinate has value x.
- Whenever the algorithms needs to process a set S_r, we reconstruct it from the stored sets, using a linear number of set operations.

To this end, we define the sets $C^i_0, C^i_1 \dots, C^i_{n_i}$ for each odd priority i. Intuitively, a vertex is in the set C^i_x iff element i of the rank of v is x. Given these $O(c+n) \in O(n)$ sets, we have encoded the exact rank vector r of each vertex with $r < \top$. To also cover vertices with rank $\top$, we additionally store the set $S_\top$.

To use the sets C^i_x, Algorithm SymbolicParityDominion has to be adapted as follows. First, at the beginning of each iteration we have to compute the set S_r and up to $c/2$ sets $S_{r'}$ that correspond to some predecessor r' of r, i.e., the sets used in line 6. Second, at the end of each iteration we have to update the sets C^i_x to incorporate the updated set S_r.

Computing a set S_r from the sets C^i_x. Let r_i denote the i-th entry of r. To obtain the set $S^=_r$ of vertices with rank *exactly* r (for $r < \top$), one can simply compute the intersection $\bigcap_{1 \le k \le c/2} C^{2k-1}_{r_{2k-1}}$ of the corresponding sets C^i_x. However, in the algorithm we need the sets S_r containing all vertices v with a rank at least r and computing all sets $S^=_{r'}$ with $r' \ge r$ is not efficient. Towards a more efficient method to compute S_r, recall that a rank $r' < \top$ is higher than r if either (a) the right most odd element of r' is larger than the corresponding element in in r, i.e., $r'_{c-1} > r_{c-1}$, or (b) if r and r' coincide on the i right most odd elements, i.e., $r'_{c-2k+1} = r_{c-2k+1}$ for $1 \le k \le i$, and $r'_{c-2i-1} > r_{c-2i-1}$. In case (a) we can compute the corresponding vertices by $S^0_r = \bigcup_{r_{c-1} < x \le n_{c-1}} C^{c-1}_x$ while in case (b) we can compute the corresponding vertices by $S^i_r = \bigcap_{1 \le k \le i} C^{c-2k+1}_{r_{c-2k+1}} \cap \bigcup_{r_{c-2i-1} < x \le n_{c-2i-1}} C^{c-2i-1}_x$ for $1 \le i \le c/2 - 1$. That is, we can reconstruct the set S_r by the following union of the above sets S_r^i, the set $S_r^=$ of vertices with rank r, and the set $S_\top$ of vertices with rank $\top$:

$$S_r = S_\top \cup S_r^= \cup \bigcup_{i=0}^{c/2-1} S_r^i$$

That is, a set S_r can be computed with $O(c+n) \in O(n)$ many $\cup$ and $O(c)$ many $\cap$ operations; for the latter bound we use an additional set to store the set $\bigcap_{1 \le k \le i} C_{r_{c-2k+1}}^{c-2k+1}$ for the current value of i, such that for each set S_r^i we just need two $\cap$ operations. This implies the following lemma.

Lemma 5.4.11. *Given the sets C_x^i as defined above, we can compute the set S_r that contains all vertices with rank at least r with $O(n)$ many symbolic set operations. No symbolic one-step operation is needed.*

Updating a set S_r. Now consider we have updated a set S_r during the iteration of the outer loop, and now we want to store the updated set S_r within the sets C_x^i. That is, we have already computed the fixed-point for S_r and are now in line 15 of the Algorithm. To this end, let S_r^{old} be the set as stored in C_x^i and S_r^{new} the updated set, which is a superset of the old one. First, one computes the difference S_r^{diff} between the two sets $S_r^{\text{diff}} = S_r^{\text{new}} \setminus S_r^{\text{old}}$; intuitively, the set S_r^{diff} contains the vertices for which the algorithm has increased the rank. Now for the vertices of S_r^{diff} we have to (i) delete their old values by updating C_x^i to $C_x^i \setminus S_r^{\text{diff}}$ for each $i \in \{1, 3, \dots, c-1\}$ and each $x \in \{0, \dots, n_i\}$ and (ii) store the new values by updating C_x^i to $C_x^i \cup S_r^{\text{diff}}$ for $i \in \{1, 3, \dots, c-1\}$ and $x = r_i$. In total we have $O(c)$ many $\cup$ and $O(n)$ many $\setminus$ operations.

Notice that the update operation for a set S_r, as described above, also updates all sets $S_{r'}$ for $r' < r$. Thus, when using the more succinct representation via the sets C_x^i and executing SymbolicParityDominion literally, the computation of the maximal rank r' s.t. $S_{r'} \supseteq S_r$ would fail because of the earlier update of S_r in line 24. Hence, we have to postpone the update of S_r till the end of the iteration (right before line 25) and adjust the computation of r' (lines 20–25) as follows. We remove line 22, i.e., we do not update the set S_r', and first compute the final value for r' by decrementing r' until $S_{r'} \supseteq S_r$ and then, before executing line 25, update S_r to S_r^{new} and thus implicitly also update the sets $S_{\tilde{r}}$ to $S_{\tilde{r}} \cup S_r$ for $r' < \tilde{r} < r$. This gives the following lemma.

Lemma 5.4.12. *In each iteration of SymbolicParityDominion only $O(n)$ symbolic set operations are needed to update the sets C_x^i, and no symbolic one-step operation is needed.*

Putting things together. To sum up, when introducing the succinct representation of the sets S_r, we only need additional $\cup$, $\cap$, and $\setminus$ operations, while the number of $CPre_z$ operations is unchanged.

In the following we argue that whenever the algorithm computes or updates a set S_r then we can charge a $CPre_z$ operation for it, and each $CPre_z$ operation is only charged for a constant number of set computations and updates. (a) Whenever the algorithm computes a set S_r in line 6 or 11, at least one $CPre_z$ computation with this set is done. (b) Now consider the computation of the new value of r. The subset tests in lines 16 and 18 are between a set that was already computed in line 6 or 11 and the set computed in line 8 or 13 and thus, if we store these sets, we do not require additional operations. Whenever a set $S_{r'}$ is considered as the left side argument in line 24, then the new value of r is less or equal to inc(r') and thus there will be another iteration of the while-loop considering inc(r'). Hence, we can charge the additional operations needed for the comparison to the $CPre_z$ operations of the next iteration that processes the rank inc(r'). (c) Finally, we only need to update the sets C_x^i once per iteration and in each iteration we perform at least one $CPre_z$ computation that we can charge for the update.

Now, as both computing a set S_r and updating the sets C_x^i can be done with $O(n)$ set operations, the number of the additional set operations in SymbolicParity-Dominion is in $O(n \cdot \#CPre)$, for $\#CPre$ being the number of $CPre_z$ operations in the algorithm. Combining Lemma 5.4.9 with the results of this section proves our key lemma.

Key Lemma 5.4.13. *For a given parity game with n vertices and c priorities and an integer $h \in [1, n-1]$, we can compute a player-$\mathcal{E}$ dominion that contains all $\mathcal{E}$-dominions with at most $h+1$ vertices with $O\left(c \cdot n \cdot \binom{h+\lfloor c/2 \rfloor}{h}\right)$ symbolic one-step operations, $O\left(c \cdot n^2 \cdot \binom{h+\lfloor c/2 \rfloor}{h}\right)$ symbolic set operations, and space for $O(n)$ many sets.*

Theorem 5.4.14. *Let $\xi(n,c) = \left(\frac{n}{\lfloor c/2 \rfloor}\right)^{\lfloor c/2 \rfloor}$. Algorithm SymbolicParityDominion correctly computes the winning sets of parity games and requires $O\big(c \cdot n \cdot \xi(n,c)\big)$ symbolic one-step operations, $O\big(c \cdot n^2 \cdot \xi(n,c)\big)$ symbolic set operations, and space for $O(n)$ many sets.*

Remark 5.4.15 (Strategy construction)**.** *Given the set representation of the progress measure returned by Algorithm SymbolicParityDominion, the winning strategy for player $\mathcal{E}$ can be computed with $O(n)$ symbolic one-step operations and $O(c \cdot n^2)$ symbolic set operations. To this end, notice that the values of the progress measure immediately give a winning strategy of player $\mathcal{E}$. That is, for a vertex v in the winning set of $\mathcal{E}$ a winning strategy of $\mathcal{E}$ picks an arbitrary successor w of v, with $\rho(w) \leq_{\alpha(v)} \rho(v)$ if v has an even priority or with $\rho(w) <_{\alpha(v)} \rho(v)$ if v has an odd priority.*

In the symbolic setting we can compute the strategy as follows. We maintain a set S of the vertices in $V_{\mathcal{E}} \setminus V_{\top}$ that already have a strategy, and first initialize S as the empty set. We then iterate over all vertices $v \in V \setminus V_{\top}$, i.e., over all vertices that are winning for player $\mathcal{E}$, and do the following.

- *First, compute the exact rank $\rho(v)$ of the vertex v by checking with $O(n)$ set operations for which of the sets C_x^i the intersection with $\{v\}$ is not empty.*[5]
- *Second, for each $0 \leq \ell < c$ compute the set $S_{\mathrm{inc}_\ell(\rho(v))}$ with $O(n)$ many symbolic set operations per set.*
- *Then compute $CPre_{\mathcal{E}}(\{v\})$ and consider the sets $CPre_{\mathcal{E}}(\{v\}) \cap S_{\mathrm{inc}_\ell(\rho(v))} \cap V_{\mathcal{E}} \cap P_\ell$, for $0 \leq \ell < c$. From each vertex contained in one of the sets a winning strategy for player $\mathcal{E}$ can move to v and thus, for each of these vertices not already in sets S, we fix v as the strategy of player $\mathcal{E}$ and add it to S.*

In each iteration of the above algorithm we use only one $CPre_z$ operation and $O(c \cdot n)$ symbolic set operations and as we have $O(n)$ iterations, the claim follows.

5.5 Symbolic Big-Step Algorithm

In the previous section we presented a set-based symbolic algorithm to compute the progress measure that is also capable of determining dominions of bounded size. Since the classical algorithm can be implemented with set-based symbolic operations, we now show how to combine our algorithm and the classical set-based algorithm to obtain a set-based symbolic *Big-Step* algorithm for parity games (see Section 5.2.6.3 for the explicit Big-Step algorithm).

Iterative winning set computation. The basic structure of Algorithm SymbolicBigStep is the same as in the classical algorithm for parity games (see Section 5.2.6.1). Let z be $\mathcal{E}$ if c is odd and $\mathcal{O}$ if c is even and assume we have $c > 2$ (the cases $c \leq 2$, i.e., Büchi games, are simpler). Let $\mathcal{G}$ be the game graph as maintained by the algorithm. In each iteration of the repeat-until loop the algorithm searches for a $\overline{z}$-dominion. When a $\overline{z}$-dominion is found, its $\overline{z}$-attractor is added to the winning set of $\overline{z}$ and removed from the game graph $\mathcal{G}$. If no $\overline{z}$-dominion is found, then the set of vertices in the remaining game graph $\mathcal{G}$ is returned as the winning set of player z.

Dominion search. The search for $\overline{z}$-dominions is conducted in two different ways: by Procedure DOMINION and by a recursive call to a derived parity game with game graph $\mathcal{G}'$ and the priority function α restricted to the vertices of $\mathcal{G}'$. The derived parity game is denoted by $(\mathcal{G}', \alpha)$ and will have $c - 1$ priorities. The parameter h is used to balance the number of symbolic operations of the two procedures.

First, all $\overline{z}$-dominions of size at most $h + 1$ are found with Procedure DOMINION that uses Algorithm SymbolicParityDominion. Note that this algorithm determines $\mathcal{E}$-dominions. We can compute $\mathcal{O}$-dominions (including a winning strategy within

[5]Alternatively we could use binary search on the sets S_r at the cost of increasing the set operations for this step by a factor of $O(c \cdot \log n)$.

Algorithm SymbolicBigStep: Symbolic Big-Step algorithm for parity games

```
Input  : parity game $P = (\mathcal{G}, \alpha)$, with
         game graph $\mathcal{G} = ((V, E), (V_{\mathcal{E}}, V_{\mathcal{O}}))$ and
         priority function $\alpha : V \to [c]$, and
         parameter $h \in [1, n] \cap \mathbb{N}$
Output: winning sets $(W_{\mathcal{E}}, W_{\mathcal{O}})$ of player $\mathcal{E}$ and player $\mathcal{O}$
if $c = 1$ then return $(V, \emptyset)$
let $z$ be player $\mathcal{E}$ if $c$ is odd and player $\mathcal{O}$ otherwise
$W_{\overline{z}} \leftarrow \emptyset$
repeat
    if $c > 2$ then
        $W'_{\overline{z}} \leftarrow$ Dominion($\mathcal{G}, \alpha, h, \overline{z}$)
        $A \leftarrow Attr_{\overline{z}}(\mathcal{G}, W'_{\overline{z}})$
        $W_{\overline{z}} \leftarrow W_{\overline{z}} \cup A$
        $\mathcal{G} \leftarrow \mathcal{G} \setminus A$
    $\mathcal{G}' \leftarrow \mathcal{G} \setminus Attr_z(\mathcal{G}, P_{c-1})$
    $(W'_{\mathcal{E}}, W'_{\mathcal{O}}) \leftarrow$SymbolicBigStep($\mathcal{G}', \alpha$)
    $A \leftarrow Attr_{\overline{z}}(\mathcal{G}, W'_{\overline{z}})$; $W_{\overline{z}} \leftarrow W_{\overline{z}} \cup A$
    $\mathcal{G} \leftarrow \mathcal{G} \setminus A$
until $W'_{\overline{z}} = \emptyset$
$W_z \leftarrow V \setminus W_{\overline{z}}$
return $(W_{\mathcal{E}}, W_{\mathcal{O}})$

Procedure Dominion($\mathcal{G}, \alpha, h, \overline{z}$)
    if $\overline{z} = \mathcal{E}$ then
        return SymbolicParityDominion for $\mathcal{G}$, $\alpha$, and $h$
    else
        construct the parity game $(\mathcal{G}', \alpha')$ from $(\mathcal{G}, \alpha)$ by increasing each priority
        by one and changing the roles of the two players
        return SymbolicParityDominion for $\mathcal{G}'$, $\alpha'$, and $h$
```

the dominion) by adding one to each priority and changing the roles of the two players. If c was even before this modification, this does not increase the bound on the number of symbolic operations for the dominion search because the number of odd priorities, and therefore the possible number of non-empty indices of a rank vector, does not increase (compare the definition of the co-domain for rank vectors M_h in Section 5.2.6.3). Whenever we call Procedure Dominion for $\overline{z} = \mathcal{O}$, we do so for a parity game where the highest priority is even; thus we can use the same upper bound on the number of symbolic operations as for $\mathcal{E}$-dominions[6].

For the recursive call for the parity game $\mathcal{P}' = (\mathcal{G}', \alpha)$ we obtain the game

[6]As for the recursive call, we could also call Procedure Dominion for a modified parity game where the highest priority vertices are removed. This would yield the same worst-case number of symbolic operations.

graph $\mathcal{G}'$ from $\mathcal{G}$ by removing the z-attractor of the vertices with the highest priority. Note that the vertices of $\mathcal{G}'$ are z-closed in $\mathcal{G}$. The winning set of player $\overline{z}$ in $\mathcal{P}'$ is a $\overline{z}$-dominion in the original parity game.

Number of iterations. The following lemma shows that the number of iterations of the repeat-until loop is bounded by $O(n/h)$. This holds because in each but the last iteration the union of the dominion identified by Dominion and the dominion identified by the recursive call is larger than $h+1$; otherwise, the union of the two dominions, which is itself a dominion, would already have been identified solely by Procedure Dominion and the algorithm would have terminated.

Lemma 5.5.1 ([Sch07]). *Let h be the parameter of Algorithm SymbolicBigStep. In each but the last iteration of the repeat-until loop at least $h+2$ vertices are removed, and thus there are at most $\lfloor n/(h+2)\rfloor + 1$ iterations.*

Proof. Consider a fixed iteration of the repeat-until loop. Let $W''_{\overline{z}}$ be the union of the $\overline{z}$-dominion identified by Procedure Dominion and the $\overline{z}$-dominion identified by the recursive call. If $|W''_{\overline{z}}| \leq h+1$, then by Lemma 5.4.9 the $\overline{z}$-dominion $W''_{\overline{z}}$ is identified by Procedure Dominion and therefore the recursive call returned the empty set and the algorithm terminates. Otherwise we have $|W''_{\overline{z}}| > h+1$, which can happen in at most $\lfloor n/(h+2)\rfloor$ iterations of the repeat-until loop since the vertices of $W''_{\overline{z}}$ are removed from the maintained game graph before the next iteration. □

Analysis of number of symbolic operations. We first analyze the number of symbolic operations of Algorithm SymbolicBigStep for instances with an arbitrary numbers of priorities c and then give better bounds for the case $c \leq \sqrt{n}$. For the former we follow the analysis of [JPZ08] for a similar explicit algorithm. There we choose the parameter h to depend (only) on the number of vertices of the game graph for which Procedure Dominion is called. In the analysis for $c \leq \sqrt{n}$ the parameter h depends on the number of vertices in the *input game graph* and the number of priorities in the game graph for which the procedure is called.

Lemma 5.5.2. *Let $h = \lceil\sqrt{2n'}\rceil - 2$ where n' is the number of vertices in $\mathcal{G}$ when Procedure* Dominion *is called in Algorithm SymbolicBigStep. With this choice for h and parity games with n vertices, Algorithm SymbolicBigStep requires $n^{O(\sqrt{n})}$ symbolic one-step and set operations and space for $O(n)$ many sets.*

Proof. Let $T(n)$ denote the number of the symbolic one-step operations of Algorithm SymbolicBigStep when called for a game graph with n vertices. We instantiate h as $h = \lceil\sqrt{2n}\rceil - 2$. First, for a game graph of size n, by Lemma 5.4.13, we can compute a dominion in $O\left(c \cdot n \cdot \binom{h+\lfloor c/2\rfloor}{h}\right)$. This can be bounded with $O(n^{h+2})$ as follows (using $c \leq n$). First we apply Stirling's approximation $(h/e)^h \leq h!$ and we distinguish whether $h \geq \lfloor c/2 \rfloor$ or $h \leq \lfloor c/2 \rfloor$. We have

$$\binom{h + \lfloor c/2 \rfloor}{h} \leq \frac{(h + \lfloor c/2 \rfloor)^h}{h!} \leq \left(\frac{(h + \lfloor c/2 \rfloor)e}{h}\right)^h ,$$

and (a) if $h \geq \lfloor c/2 \rfloor$, then

$$\left(\frac{(h + \lfloor c/2 \rfloor)e}{h}\right)^h \leq \left(\frac{2h \cdot e}{h}\right)^h \leq (2e)^h ,$$

(b) if $h \leq \lfloor c/2 \rfloor$, then

$$\left(\frac{(h + \lfloor c/2 \rfloor)e}{h}\right)^h \leq \left(\frac{2\lfloor c/2 \rfloor \cdot e}{h}\right)^h .$$

In both cases we obtain $O(n^{h+2})$.

Let $\Gamma = h+2 = \lceil \sqrt{2n} \rceil$. To solve an instance of size n, the algorithm (1) calls the subroutine for computing a dominion, which is in $O(n^{\Gamma})$; (2) then makes a recursive call to Algorithm SymbolicBigStep, with an instance of size at most size $n - 1$ (it removes the largest parity and thus at least one vertex); and (3) finally processes the instance in which all winning vertices from the previous two steps have been removed, i.e., an instance of size at most $n - \Gamma$ (Lemma 5.5.1). Thus, we obtain the following recurrence relation for $T(n)$

$$T(n) \leq O(n^{\Gamma}) + T(n-1) + T(n-\Gamma)$$

for $\Gamma = \lceil \sqrt{2n} \rceil$. This coincides with the recurrence relation for the explicit algorithm and thus we get $T(n) = n^{O(\sqrt{n})}$ [JPZ08, Theorem 8.1]. Finally, as in Algorithm SymbolicBigStep the number of symbolic set operations is at most a factor of n higher than the number of symbolic one-step operations, we obtain the claim by $n \cdot T(n) = n^{O(\sqrt{n})}$.

Now let us consider the space requirements of Algorithm SymbolicBigStep. Computing a dominion, by Lemma 5.4.13, can be done with $O(n)$ many sets. The recursion depth is bounded by c and thus also by n, and for each instance on the stack we just need a constant number of sets to store the current subgraph and the winning sets of the two players. Finally, also attractor computations can be done with a constant number of sets. That is, Algorithm SymbolicBigStep uses $O(n)$ many sets, independent of the value of h. □

To provide an improved bound for $c \leq \sqrt{n}$ we follow the analysis of [Sch07]; see [Sch17] (in press) for an improved analysis. We use the following definitions from [Sch07]: let $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c, and let $\beta(c) = \gamma(c)/(\lfloor c/2 \rfloor + 1)$. With these definitions we have $\beta(c) \in [1/2, 7/10]$ for all $c \geq 3$, $\gamma(c) \in [c/3, c/3 + 1/2]$ and

$$\gamma(c) = \gamma(c-1) + 1 - \beta(c-1) , \tag{5.1}$$

$$\beta(c-1) \cdot \lceil c/2 \rceil = \gamma(c-1) . \tag{5.2}$$

We first present a shorter proof for the simplified bound of $O(n^{1+\gamma(c)})$ symbolic one-step operations and then a more extensive analysis to show that, for some constant κ, $O(n \cdot (\kappa \cdot n/c)^{\gamma(c)})$ symbolic one-step operations are sufficient. We bound the number of symbolic set operations with an additional factor of $O(n)$. In both cases the parameter h depends on number of vertices in the input game graph and the number of priorities c in the parity game that is maintained by the algorithm.

Lemma 5.5.3 (Simplified analysis of number of symbolic operations)**.** *For parity games with n vertices and c priorities Algorithm SymbolicBigStep takes $O(n^{1+\gamma(c)})$ symbolic one-step operations and $O(n^{2+\gamma(c)})$ symbolic set operations for $2 < c \leq \sqrt{n}$ and $O(n^2)$ symbolic one-step and set operations for $c = 2$. Moreover, it needs to store $O(n)$ many sets.*

Proof. By the proof of Lemma 5.5.2 the algorithm only stores $O(n)$ many sets independent of the value of h, and thus it only remains to show the upper bound on the number of symbolic operations. We focus on the bound for the number of symbolic one-step operations, the number of symbolic set operations is higher only for the subroutine SymbolicParityDominion, where the difference is at most a factor of $O(n)$, by Lemma 5.4.9. For $c = 2$ note that in each iteration of the repeat-until loop at least one vertex is removed from the game, thus there can be at most $O(n)$ iterations. In each iteration there are attractor computations, a recursive call for parity games with one priority, and set operations, which all can be done with $O(n)$ symbolic operations[7], thus at most $O(n^2)$ symbolic operations are needed for $c = 2$.

By Lemma 5.5.1 the repeat-until loop can have at most $\left\lfloor \frac{n}{(h+2)} \right\rfloor + 1$ iterations. We show the claimed number of symbolic one-step operations for $c \geq 3$ by induction over c. For the base case of $c = 3$ let $h = n$. In this case there is only one iteration of the repeat-until loop. The call to the progress measure procedure and the recursive call for parity games with one priority less, i.e., Büchi games, both take $O(n^2)$ symbolic one-step operations. Thus the total number of symbolic one-step operations for $c = 3$ is bounded by $O(n^{1+\gamma(3)}) = O(n^2)$. For $c = 3$ the number of sets S_r is only $O(n)$, therefore we do not have to use the sets C_x^i to achieve linear space and thus the number of symbolic set operations is also $O(n^2)$.

For the induction step we differentiate between the number of vertices n_0 in the first (i.e., non-recursive) call to Algorithm SymbolicBigStep and the number of vertices n' in the parity game that is maintained by the algorithm. We set the parameter h according to n_0 (and to at most n') to maintain the property $c \leq \sqrt{n_0}$ in all recursive calls.

Let $h = \min\left(\lceil n_0^{\beta(c-1)} \rceil, n'\right)$ for $c > 3$. By Lemma 5.5.1 the number of iterations in Algorithm SymbolicBigStep is bounded by $O(n_0^{1-\beta(c-1)})$. By Lemma 5.4.9 each

[7]Note that while the attractor computations in line 12 only need $O(n)$ symbolic operations over the whole algorithm, the attractor computations in line 10 might require $O(n)$ symbolic operations in each iteration.

call to SymbolicParityDominion needs only

$$O\left(c \cdot n_0 \cdot \binom{\lceil n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor}{\lceil n_0^{\beta(c-1)} \rceil}\right)$$

many symbolic one-step operations. The binomial coefficient can be bounded using $\beta(c-1) \geq 1/2$, Stirling's approximation of $(c/e)^c \leq c!$, and $3 < c \leq \sqrt{n_0}$.

$$\begin{aligned}
\binom{\lceil n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor}{\lceil n_0^{\beta(c-1)} \rceil} &\leq \frac{(\lceil n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor)^{\lfloor c/2 \rfloor}}{\lfloor c/2 \rfloor !}, \\
&\leq \left(\frac{2e \cdot (\lceil n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor)}{c-1}\right)^{\lfloor c/2 \rfloor}, \\
&\leq \left(\frac{2e \cdot n_0^{\beta(c-1)} + e \cdot c}{c-1}\right)^{\lfloor c/2 \rfloor}, \\
&\leq \left(\frac{4e \cdot n_0^{\beta(c-1)}}{c}\right)^{\lfloor c/2 \rfloor},
\end{aligned}$$

thus, the number of symbolic one-step operations in a call to SymbolicParityDominion is bounded by $O(n_0 \cdot n_0^{\beta(c-1)\lfloor c/2 \rfloor})$. Hence, we can bound the total number of symbolic one-step operations for all calls to SymbolicParityDominion with $O(n_0 \cdot n_0^{\beta(c-1)\lfloor c/2 \rfloor} \cdot n_0^{1-\beta(c-1)})$. Using Eq. (5.2), we have $\beta(c-1)\lfloor c/2 \rfloor \leq \beta(c-1)\lceil c/2 \rceil = \gamma(c-1)$ and can bound the total number of symbolic one-step operations of the calls to the progress measure procedure with $O(n_0 \cdot n_0^{\gamma(c-1)+1-\beta(c-1)})$, which by Eq. (5.1) is equal to $O(n_0^{1+\gamma(c)})$. By the induction assumption we further have that the total number of symbolic one-step operations for all recursive calls is $O(n_0^{1-\beta(c-1)+1+\gamma(c-1)}) = O(n_0^{1+\gamma(c)})$, which concludes the proof. □

Lemma 5.5.4 (Number of symbolic operations). *For parity games with n vertices and $2 < c \leq \sqrt{n}$ priorities Algorithm SymbolicBigStep takes, for some constant κ, $O\big(n \cdot (\frac{\kappa \cdot n}{c})^{\gamma(c)}\big)$ symbolic one-step operations and $O\big(n^2 \cdot (\frac{\kappa \cdot n}{c})^{\gamma(c)}\big)$ symbolic set operations. Moreover, it needs to store $O(n)$ many sets.*

Proof. By the proof of Lemma 5.5.2 the algorithm only stores $O(n)$ many sets independent of the value of h, and thus it only remains to show the upper bound on the number of symbolic operations. We show the bound for the number of symbolic one-step operations, the number of symbolic set operations is higher only for the subroutine SymbolicParityDominion, where the difference is at most a factor of $O(n)$ by Lemma 5.4.9.

We differentiate between the number of vertices n_0 in the first (i.e. non-recursive) call to Algorithm SymbolicBigStep and the number of vertices n' in the parity game that is maintained by the algorithm. We will set the parameter h according to n_0 (and to at most n') to maintain the property $c \leq \sqrt{n_0}$ in all recursive calls.

Similar to [Sch08], we use $h = \min\left(\lceil 2\sqrt[3]{c} \cdot n_0^{\beta(c-1)} \rceil, n'\right)$ to balance the number of symbolic operations to compute dominions with the number of symbolic operations in the recursive calls. We will frequently apply *Stirling's approximation* by which we have

(S1) $\left(\frac{c}{e}\right)^c \leq c!$, and

(S2) $c! \in O\left(\left(\frac{c}{\hat{\kappa}}\right)^c\right)$ for all $\hat{\kappa} < e$.

We first show that proving the following claim is sufficient.

Claim 5.5.5. *The number of symbolic one-step operations is bounded by $\kappa_1 \cdot n_0 \cdot \frac{(\kappa_2 \cdot n_0)^{\gamma(c)}}{\sqrt[3]{c!}}$ for some constants κ_1 and κ_2.*

The claim implies the lemma since we have for $c \geq 3$

$$\begin{aligned}
\kappa_1 \cdot n_0 \cdot \frac{(\kappa_2 \cdot n_0)^{\gamma(c)}}{\sqrt[3]{c!}} &\leq \kappa_1 \cdot n_0 \cdot (\kappa_2 \cdot n_0)^{\gamma(c)} \cdot \left(\frac{e}{c}\right)^{\frac{c}{3}}, \\
&\leq \kappa_1 \cdot n_0 \cdot (\kappa_2 \cdot n_0)^{\gamma(c)} \cdot \left(\frac{e}{c}\right)^{\gamma(c)-\frac{1}{2}}, \\
&\leq \kappa_1 \cdot n_0 \cdot \left(\frac{\kappa_2 \cdot e \cdot \kappa_3 \cdot n_0}{c}\right)^{\gamma(c)}
\end{aligned}$$

for some constant $\kappa_3 > 1$. The first inequality is by (S1) and the second by the definition of $\gamma(c)$. Thus it remains to prove the above claim.

We prove the claim by induction over c, where the base case is provided by Lemma 5.5.3 for, e.g., $c = 3$. Assume the claim holds for parity games with $c - 1$ priorities; we will show that the claim then also holds for parity games with c priorities. For $c > 2$ the number of symbolic operations per iteration is dominated by (a) the recursive call for parity games with one priority less and (b) the call to Procedure Dominion. To bound the total number of symbolic operations for (a) and (b), we use that by Lemma 5.5.1 the number of iterations in Algorithm SymbolicBigStep is bounded by

$$\left\lfloor \frac{n_0}{h+2} \right\rfloor + 1 \leq \frac{n_0^{1-\beta(c-1)}}{2\sqrt[3]{c}} + 1 \leq 2 \cdot \frac{n_0^{1-\beta(c-1)}}{3\sqrt[3]{c}}.$$

(a) We first bound the total number of symbolic one-step operations for the recursive call. By the induction hypothesis the number of symbolic one-step operations in a single recursive call is bounded by

$$\kappa_1 \cdot n_0 \cdot \frac{(\kappa_2 \cdot n_0)^{\gamma(c-1)}}{\sqrt[3]{(c-1)!}}.$$

By the definition of $\gamma(c)$ (see also the proof of Lemma 5.5.3) we can bound the total number of symbolic one-step operations for all recursive calls with

$$\kappa_1 \cdot n_0 \cdot \frac{(\kappa_2 \cdot n_0)^{\gamma(c-1)}}{\sqrt[3]{(c-1)!}} \cdot \frac{2 \cdot n_0^{1-\beta(c-1)}}{3\sqrt[3]{c}} \leq \kappa_1 \cdot n_0 \cdot \frac{2 \cdot (\kappa_2 \cdot n_0)^{\gamma(c)}}{3\sqrt[3]{c!}} . \tag{5.3}$$

(b) Thus it remains to bound the total number of symbolic one-step operations in the calls to Procedure Dominion. By Lemma 5.4.9 the number of symbolic one-step operations in a call to Procedure Dominion is bounded by $O\left(c \cdot n_0 \cdot \binom{h+\lfloor c/2 \rfloor}{\lfloor c/2 \rfloor}\right)$. The binomial coefficient can be bounded by using $\beta(c-1) \geq 1/2$, Stirling's approximation (S1), and $3 < c \leq \sqrt{n_0}$. We have

$$\binom{\lceil 2\sqrt[3]{c} \cdot n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor}{\lfloor c/2 \rfloor} \leq \frac{(\lceil 2\sqrt[3]{c} \cdot n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor)^{\lfloor c/2 \rfloor}}{\lfloor c/2 \rfloor !},$$
$$\leq \left(\frac{e \cdot (\lceil 2\sqrt[3]{c} \cdot n_0^{\beta(c-1)} \rceil + \lfloor c/2 \rfloor)}{\lfloor c/2 \rfloor}\right)^{\lfloor c/2 \rfloor} \leq \left(\frac{(1+1/4) \cdot e \cdot \lceil 2\sqrt[3]{c} \cdot n_0^{\beta(c-1)} \rceil}{\lceil (c-1)/2 \rceil}\right)^{\lceil \frac{c-1}{2} \rceil} .$$

With this bound on the binomial coefficient we have that the number of symbolic one-step operations in one call to Procedure Dominion is bounded by

$$\kappa_4 \cdot c \cdot n_0 \cdot \left(\frac{7\lceil 2\sqrt[3]{c} n_0^{\beta(c-1)} \rceil}{c-1}\right)^{\lceil \frac{c-1}{2} \rceil}$$

for some constant κ_4. To obtain a bound on the total number of symbolic one-step operations for all calls to Procedure Dominion, we multiply that with the number of iterations $2 \cdot n_0^{1-\beta(c-1)}/(3\sqrt[3]{c})$ and split the further analysis into two parts.
(1) First, we consider the factor $\lceil n_0^{\beta(c-1)} \rceil^{\lceil \frac{c-1}{2} \rceil} \cdot n_0^{1-\beta(c-1)}$ depending on n_0. We upper bound the factor by using $c \leq \sqrt{n_0}$ and $\beta(c-1) \geq 1/2$, and then apply Eq. (5.1).

$$\lceil n_0^{\beta(c-1)} \rceil^{\lceil \frac{c-1}{2} \rceil} \cdot n_0^{1-\beta(c-1)} \leq \sqrt{e} \cdot n_0^{\beta(c-1) \cdot \lceil \frac{c}{2} \rceil} \cdot n_0^{1-\beta(c-1)} = \sqrt{e} \cdot n_0^{\gamma(c)} .$$

(2) Second, we consider the factor $c \cdot \left(7\lceil 2\sqrt[3]{c} \rceil/(c-1)\right)^{\lceil \frac{c-1}{2} \rceil}$ depending on c and show that, when assuming[8] $c \geq 64$, we have

$$c \cdot \left(\frac{7\lceil 2\sqrt[3]{c} \rceil}{c-1}\right)^{\lceil \frac{c-1}{2} \rceil} \leq c \cdot \left(\frac{\kappa_5 \sqrt[3]{c}}{c-1}\right)^{\frac{c-1}{2}},$$
$$\leq c \cdot \left(\frac{\kappa_6 \sqrt[3]{c-1}}{c-1}\right)^{\frac{c-1}{2}},$$

[8]Notice that for $c \leq 63$ the factor is bounded by a constant anyway.

$$\leq c \cdot \left(\frac{\sqrt{\kappa_6}}{\sqrt[3]{c-1}} \right)^{c-1},$$
$$\leq c \cdot \left(\frac{4}{\sqrt[3]{c-1}} \right)^{c-1}.$$

with $\kappa_5 = 63/4$, and $\kappa_6 = \kappa_5 \cdot \sqrt[3]{64/63} \approx 15.83$. Notice that in the above equation we exploited the assumption $c \geq 64$ several times. First, to bound $7\lceil 2\sqrt[3]{c}\rceil$ by $\kappa_5\sqrt[3]{c}$, second, to obtain $\left(\frac{\kappa_5\sqrt[3]{c}}{c-1}\right) \leq 1$, and, finally, to bound $\kappa_5\sqrt[3]{c}$ by $\kappa_6\sqrt[3]{c-1}$.

Now let κ_7 be such that $\kappa_7/(c-1)! \geq (\hat{\kappa}/(c-1))^{c-1}$ for some $\hat{\kappa}$ slightly smaller than e, such that we can apply (S2). Notice that we also can bound $4^3/\hat{\kappa}$ by 24 for $\hat{\kappa}$ close to e. We have

$$c \cdot \left(\frac{4 \cdot \sqrt[3]{\hat{\kappa}}}{\sqrt[3]{\hat{\kappa}} \cdot \sqrt[3]{(c-1}} \right)^{c-1} \leq c \cdot \frac{\sqrt[3]{\kappa_7} \cdot 24^{(c-1)/3}}{\sqrt[3]{(c-1)!}},$$
$$\leq \frac{\kappa_8 \cdot \kappa_9^{\gamma(c)}}{\sqrt[3]{(c-1)!}},$$

for appropriately chosen constants κ_8 and κ_9. Putting parts (1), (2) and the not yet considered factor of $\kappa_4 \cdot 2/(3\sqrt[3]{c})$ together, we have that the total number of symbolic one-step operations for all calls to Procedure DOMINION is bounded by

$$\kappa_4 \cdot c \cdot n_0 \cdot \left(\frac{7\lceil 2\sqrt[3]{cn_0^{\beta(c-1)}}\rceil}{\lceil\frac{c-1}{2}\rceil} \right)^{\lceil\frac{c-1}{2}\rceil} \cdot \frac{2 \cdot n_0^{1-\beta(c-1)}}{3\sqrt[3]{c}} \leq \kappa_1 \cdot n_0 \cdot \frac{(\kappa_2 \cdot n_0)^{\gamma(c)}}{3\sqrt[3]{c!}}$$

for $\kappa_1 \geq 2\sqrt{e} \cdot \kappa_4 \cdot \kappa_8$ and $\kappa_2 \geq \kappa_9$. Together with Equation (5.3) this concludes the proof. □

Correctness. The correctness of Algorithm SymbolicBigStep is as follows: (i) the algorithm correctly identifies dominions for player $\overline{z}$ (Lemma 5.4.13), and then removes them (Lemma 2.5.1 (3)), and it suffices to solve the remaining game graph; and (ii) when no vertices are removed, then no dominion is found, and the computation is exactly as the classical algorithm, and from the correctness of the classical algorithm it follows that the remaining vertices are winning for player z.

Lemma 5.5.6 (Correctness). *The algorithm correctly computes the winning sets of player $\mathcal{E}$ and player $\mathcal{O}$ for parity games with $c \geq 1$ priorities.*

Proof. The proof is by induction over c. For $c = 1$ all vertices are winning for player $\mathcal{E}$ and the algorithm correctly returns $W_{\mathcal{E}} = V$ and $W_{\mathcal{O}} = \emptyset$. Assume the algorithm correctly computes the winning sets for parity games with $c-1$ priorities. We show that this implies the correctness for c priorities. Let z be $\mathcal{E}$ if c is odd and

$\mathcal{O}$ otherwise. We first show (1) that each vertex of $W_{\overline{z}}$ is indeed winning for player $\overline{z}$ and then (2) that each vertex of W_z is winning for player z; this is sufficient to prove the lemma by $W_{\overline{z}} \cup W_z = V$.

For (1) recall that the algorithm repeatedly computes $\overline{z}$-dominions D and their $\overline{z}$-attractor A, adds A to the winning set of player $\overline{z}$ and recurses on the parity game with A removed. By Lemma 2.5.1 (3) we have that this approach is correct if the sets D are indeed $\overline{z}$-dominions. By the soundness of SymbolicParityDominion by Lemma 5.4.13 we have that D is a $\overline{z}$-dominion when computed with SymbolicParityDominion; it remains to show the soundness of determining a dominion $W'_{\overline{z}}$ of player $\overline{z}$ by the recursive call to SymbolicBigStep on the parity game $(\mathcal{G}', \alpha)$ with one priority less (line 11). This follows by Lemma 2.5.1 (1) from $\mathcal{G}'$ not containing a vertex with priority $c - 1$ and being closed for player $\mathcal{E}$ by Lemma 2.5.1 (1).

To show completeness, i.e., $W_{\overline{z}}(P) \subseteq W_{\overline{z}}$, we describe a winning strategy for player z on the vertices of W_z. Since the set W_z is the set of remaining vertices after the removal of $\overline{z}$-attractors, the set W_z is $\overline{z}$-closed by Lemma 2.5.1 (1). Let $\mathcal{G}^*$ be the game graph as in the last iteration of the algorithm, i.e., $\mathcal{G}^* = \mathcal{G}[W_z]$. Furthermore, let Z be the set of vertices in $\mathcal{G}^*$ with priority $c - 1$, and let $\mathcal{G}' = \mathcal{G}^* \setminus Attr_z(\mathcal{G}^*, Z)$. Since the algorithm has terminated, we have that player z wins on all vertices of $\mathcal{G}'$ in the parity game $(\mathcal{G}', \alpha)$. The winning strategy for player z for the vertices of W_z is as follows: for vertices of Z the player choses an edge to some vertex in W_z; for vertices of $Attr_z(\mathcal{G}^*, Z) \setminus Z$ the player follows her attractor strategy to Z; and for the vertices of $\mathcal{G}'$ the player plays according to her winning strategy in $(\mathcal{G}', \alpha)$. Then in a play starting from W_z either Z is visited infinitely often or the play remains within $\mathcal{G}'$ from some point on; in both cases player z wins with the given strategy. □

With Lemmata 5.5.6, 5.5.2, and 5.5.4 we proved the following result.

Theorem 5.5.7. *Let $\gamma(c) = c/3 + 1/2 - 4/(c^2 - 1)$ for odd c and $\gamma(c) = c/3 + 1/2 - 1/(3c) - 4/c^2$ for even c. Algorithm SymbolicBigStep correctly computes the winning sets for parity games and requires the minimum of $O(n \cdot (\kappa \cdot n/c)^{\gamma(c)})$ for some constant κ and $n^{O(\sqrt{n})}$ symbolic one-step operations and space to store $O(n)$ many sets. The number of symbolic set operations is at most a factor of $O(n)$ larger than the number of one-step operations.*

Remark 5.5.8 (Strategy construction)**.** *The winning strategies for both players can be computed within the bounds of Theorem 5.5.7 using SymbolicBigStep. This is by the following observations.*

- *Whenever a player-$\overline{z}$ dominion is found with SymbolicParityDominion (line 6), then by Remark 5.4.15 we can obtain a winning strategy of player $\overline{z}$ for each vertex in the $\overline{z}$-dominion within the same algorithmic bounds.*
- *For $c = 1$ player $\mathcal{E}$ can just pick any successor vertex and thus a strategy can be constructed with $O(|V|)$ many symbolic operations.*

- *For $\overline{z}$-dominion found in a recursive call to SymbolicParityDominion (line 11), we can use the strategy that is computed in the recursive call.*
- *For vertices in the $\overline{z}$-attractor A computed in lines 7 and 12, we can determine the winning strategy with $O(|A|)$ symbolic operations as follows. In the computation of the attractor we keep track of the vertices A_{i-1} added in the previous iteration of the attractor computation and whenever a vertices of $V_{\overline{z}}$ is added to the attractor, we identify a successor for each of them. To this end, for each vertex in $v \in A_{i-1}$, we compute $Pre(\{v\}) \cap A_i \cap V_{\overline{z}}$ and fix v as players $\overline{z}$ choice for all these vertices. As each vertex of A is in exactly one set A_i, we only need $O(|A|)$ many operations, and as the attractor computation itself needs $O(|A|)$ many $CPre_z$ operations, this does not increase the bound for the number of symbolic operations.*

5.6 Conclusion

In this section we presented the first sub-cubic time (explicit) algorithm for parity-3 games and improved set-based symbolic algorithms for parity games. The bounds on the number of symbolic operations and the space usage (in terms of the number of stored sets) match the known bounds on the running time and space, respectively, of explicit algorithms for parity games. It would be interesting to understand the relation between the two models better. For both models the major and long-standing theoretical open problem is finding a polynomial-time algorithm. To this end, conditional lower bounds could provide more insights into the complexity of parity games and its relation to other combinatorial problems. Another interesting direction of future work would be to explore a practical approach, where our algorithmic ideas are complemented with engineering efforts to obtain scalable symbolic algorithms for the reactive synthesis of systems.

CHAPTER 6

Model and Objective Separation for Graphs and MDPs

6.1 Introduction

In this chapter we present improved algorithms and the first super-linear lower bounds for polynomial-time algorithmic problems in model checking on graphs and MDPs. We consider several ω-regular objectives, in particular reachability, safety, Büchi, and co-Büchi objectives and conjunctions and disjunctions thereof, including Streett and Rabin objectives. Additional motivation can be found in Chapter 1, formal definitions are provided in Chapter 2.

Significance of model and objectives. Standard graphs are the model for non-deterministic systems, and provide the framework to model hardware and software systems [Hol97; Cim+00], as well as many basic logic-related questions such as automata emptiness. MDPs model systems with both non-deterministic and probabilistic behavior; and provide the framework for a wide range of applications from randomized communication and security protocols, to stochastic distributed systems, to biological systems [KNP11; BK08]. In verification, reachability objectives are the most basic objectives for safety-critical systems. In general all properties that arise in verification (such as liveness, fairness) are ω-regular languages, and every ω-regular language can be expressed as a Streett objective (or a Rabin objective). Important special cases of Streett (resp. Rabin) objectives are Büchi and coBüchi objectives [CH14]. Thus the algorithmic questions we consider are the most basic questions in formal verification.

Conjunction and disjunction of objectives. Conjunctive and disjunctive objectives are a natural extension of single objectives [FH10; Wol00; CHP07]. For two objectives ϕ_1 and ϕ_2 their conjunctive objective is equal to $\phi_1 \cap \phi_2$ and their disjunctive objective is equal to $\phi_1 \cup \phi_2$ (see also Section 2.3). The conjunction of reachability (resp. Büchi) objectives is known as *generalized reachability* (*resp. Büchi*) [FH10; Wol00]. In addition to computing (almost-sure) winning sets and strategies for given models and objectives (see Section 2.4), we consider two additional *algorithmic questions*: The *conjunctive query* question for k objectives asks whether there is a policy for player 1 to ensure that *all* the objectives are satisfied with probability 1, and the *disjunctive query* question asks whether there is a policy for player 1 to ensure that *one* of the objectives is satisfied with probability 1. Conjunctive queries coincide with conjunctive objectives on graphs and MDPs, while disjunctive queries coincide with disjunctive objectives on graphs but not on MDPs (see Observations 6.2.1 and 6.2.2).

Conditional lower bounds. For the problems we consider, while polynomial-time algorithms are known, there are no super-linear lower bounds. Since for polynomial-time algorithms unconditional super-linear lower bounds are very rare in the whole of computer science, we consider *conditional lower bounds*, which assume that for some well-studied problem the known algorithms are optimal up to some lower-order factors. We consider two such well-studied assumptions, (A1) the combinatorial Boolean matrix multiplication conjecture (BMM) and (A2) the Strong Exponential Time Hypothesis (SETH). See Section 2.7 for a formal statement of the conjectures and Chapter 1 for their relevance.

Model separation and objective separation questions. In this chapter our results (upper and conditional lower bounds) aim to establish the following two fundamental separations:

- *Model separation.* Consider an objective where the algorithmic question for both graphs and MDPs can be solved in polynomial time, and establish a conditional lower bound for the running time for MDPs that is strictly higher than the best-known upper bound for graphs. In other words, the conditional lower bound would separate the asymptotic running times of the models of graphs and MDPs for problems (i.e., w.r.t. the objective) that can be solved in polynomial time.

- *Objective separation.* Consider a model (either graphs or MDPs) with two different objectives and show that, though the algorithmic question for both objectives can be solved in polynomial time, there is a conditional lower bound for one objective that is strictly higher than the best-known upper bound for the other objective. In other words, the conditional lower bound would separate the two objectives w.r.t. the model, though they both can be solved in polynomial time.

To the best of our knowledge, there is no previous work that establish any model separation or objective separation result in the literature.

Our results. In this chapter we present improved algorithms as well as the first conditional lower bounds that are super-linear for algorithmic problems in model checking that can be solved in polynomial time, and together they establish both model separation and objective separation results. The algorithms for Streett objectives are presented in Chapter 4. An overview of the results for the different objectives is given in Table 6.1, where our results are highlighted in boldface. We use MEC to refer to the time to compute all maximal end-components (MECs) of an MDP. Recall from Section 2.5.2 that an end-component is a non-trivial strongly connected sub-MDP (induced by a set of vertices), and that a sub-MDP has no outgoing random edges. We have $\text{MEC} = O(\min(n^2, m^{1.5}))$ [CH14] and assume $\text{MEC} = \Omega(m)$ and $m \geq n$. Moreover, we use k to denote the number of combined objectives in the case of conjunction or disjunction of multiple objectives and b to denote the total number of elements in all the target sets that specify the objectives. We first describe Table 6.1 and our main results and then discuss the significance of our results for model and objective separation.

(1) *Conjunctive and Disjunctive Reachability* (*and Büchi*) *Problems.* First, we consider conjunctive and disjunctive reachability objectives and queries. Recall that conjunctive objectives and queries in general and disjunctive objectives and queries on graphs coincide. For reachability further the disjunctive objective can be reduced to a single objective (see Observation 6.2.3). The following results are known: the algorithmic question for conjunctive reachability objectives is NP-complete for graphs [CAM13], and $PSPACE$-complete for MDPs [FH10]; and the disjunctive objective can be solved in linear time for graphs and in $O(\min(n^2, m^{1.5}) + b)$ time for MDPs [CJH03; CH14]. We present three results for disjunctive reachability queries on MDPs: (i) We present an $O(km + \text{MEC})$ time algorithm[1]. (ii) We show that under assumption (A1) there does not exist a combinatorial $O(k \cdot n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$. (iii) We show that for $k = \Omega(m)$ there does not exist an $O(m^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ under assumption (A2). Hence we establish an upper bound and matching conditional lower bounds based on (A1) and (A2).

Disjunctive Büchi objectives (on graphs and MDPs) can be reduced in linear time to disjunctive reachability objectives and vice versa, therefore the same results apply to disjunctive Büchi problems (see Observation 6.2.6). The basic algorithm for conjunctive Büchi objectives runs in time $O(m + b)$ for graphs and in time $O(\text{MEC} + b)$ for MDPs.

(2) *Conjunctive and Disjunctive Safety Problems.* Second, we consider conjunctive and disjunctive safety objectives and queries. The following results are known: the conjunctive problem can be reduced to a single objective and can be solved in linear time, both in graphs and MDPs (see e.g. [CDH10]); disjunctive queries for MDPs can be solved in $O(k \cdot m)$ time; and disjunctive

[1]This implies an $O(\text{MEC} + b)$ time algorithm for disjunctive reachability objectives but does not improve the running time for this case.

Table 6.1: Upper and lower bounds. Our results are boldface, respective results are referred, and the remaining results are folklore.

		Graphs		MDPs	
		upper bound	lower bound*	upper bound	lower bound*
Reach	Conj.	NP-c [CAM13]		$PSPACE$-c [FH10]	
	Disj. Obj.	$\Theta(m+b)$		$O(\textsc{mec}+b)$ [CJH03; CH14]	
	Disj. Qu.			$\boldsymbol{O(k \cdot m + \mathrm{MEC})}$ [Th. 6.3.1]	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.3.9], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.3.12]
Safety	Conj.	$\Theta(m+b)$		$\Theta(m+b)$	
	Disj. Obj.	$O(k \cdot m)$	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.4.2]	$PSPACE$-c [FH10]	
	Disj. Qu.			$O(k \cdot m)$	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.4.2], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.4.5]
Büchi	Conj.	$\Theta(m+b)$		$O(\textsc{mec}+b)$	
	Disj. Obj.	$\Theta(m+b)$		$O(\textsc{mec}+b)$ [CJH03; CH14]	
	Disj. Qu.			$\boldsymbol{O(k \cdot m + \mathrm{MEC})}$ [Th. 6.3.1]	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]
co-Büchi	Conj.	$\Theta(m+b)$		$O(\textsc{mec}+b)$	
	Disj. Obj.	$O(k \cdot m)$	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1]	$\boldsymbol{O(k \cdot m + \mathrm{MEC})}$ [Th. 6.5.3]	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]
	Disj. Qu.			$\boldsymbol{O(k \cdot m + \mathrm{MEC})}$ [Th. 6.5.3]	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]
Singleton	Disj. Obj.	$\boldsymbol{\Theta(m)}$ [Th. 6.6.1]			$\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]
	Disj. Qu.				$\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]
Streett		$O(\min(\boldsymbol{n^2}, m^{1.5}\sqrt{\log n}, km) + b \log n)$ [Th. 4.1.1], [HT96]		$\boldsymbol{O(\min(n^2, m^{1.5}\sqrt{\log n}) + b \log n)}$ [Th. 4.1.1]	
Rabin		$O(k \cdot m)$	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1]	$O(k \cdot \textsc{mec})$	$\boldsymbol{k \cdot n^{2-o(1)}}$ [Th. 6.5.1], $\boldsymbol{m^{2-o(1)}}$ [Th. 6.5.2]

* $\boldsymbol{k \cdot n^{2-o(1)}}$ lower bounds are based on the Combinatorial BMM Conjecture / Combinatorial Triangle Conjecture (A1)
$\boldsymbol{m^{2-o(1)}}$ lower bounds are based on the Orthogonal Vectors Conjecture / Strong ETH (A2)

Table 6.2: Model separation.

	upper bound graphs	lower bounds MDPs
Reach/Büchi Disj. Qu.	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}, \boldsymbol{m^{2-o(1)}}$
co-Büchi Singleton Disj. Obj./Qu.	$\boldsymbol{m}$	$\boldsymbol{m^{2-o(1)}}$

objectives for MDPs are $PSPACE$-complete [FH10]. We present two results: (i) We show that for the disjunctive problem in graphs under assumption (A1) there does not exist a combinatorial $O(k \cdot n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$. This implies the same conditional lower bound for disjunctive queries and objectives in MDPs and matches the upper bound for graphs and disjunctive queries in MDPs. (ii) We present, for $k = \Omega(m)$, an $\Omega(m^{2-o(1)})$ lower bound for disjunctive objectives and queries in MDPs under assumption (A2). Again this lower bound matches the upper bound of $O(k \cdot m)$ for disjunctive queries.

(3) *Conjunctive and Disjunctive co-Büchi Problems.* For co-Büchi, a conjunctive objective can be reduced to a single objective. For single objectives the basic algorithm runs in time $O(\textsc{mec} + b)$ for MDPs and in time $O(m + b)$ for graphs. Since the conditional lower bounds for disjunctive safety objectives and queries actually already apply for the non-emptiness of the winning set, the reductions also hold for co-Büchi (see Observation 6.2.5). Here the running times and the conditional lower bounds are matching for both disjunctive queries and disjunctive objectives. For the conditional lower bound based on assumption (A2) only *singleton co-Büchi* objectives, i.e., co-Büchi objectives with target sets of cardinality one, are needed, therefore the bound already holds for this case. We additionally present two results: (i) We present $O(km + \textsc{mec})$ time algorithms for disjunctive queries and objectives on MDPs. (ii) We present a linear time algorithm for disjunctive singleton co-Büchi objectives on graphs.

(4) *Rabin and Streett objectives.* Finally, we consider Rabin and Streett objectives. The basic algorithm for Rabin objectives runs in time $O(k \cdot m)$ on graphs and in time $O(k \cdot \textsc{mec})$ on MDPs. As disjunctive co-Büchi objectives are a special case of Rabin objectives, the conditional lower bounds for co-Büchi objectives of $\Omega(k \cdot n^{2-o(1)})$ for graphs and additionally $\Omega(m^{2-o(1)})$ for MDPs extend to Rabin objectives. The conditional lower bound for graphs is matching (for combinatorial algorithms). In Chapter 4 we show that MDPs with Streett objectives can be solved in $O(m^{1.5}\sqrt{\log n} + b \log n)$ and in $O(n^2 + b \log n)$ time. The first bound is an extension of an existing algorithm for graphs [HT96], while the latter is also an improvement for graphs with $m > n^{4/3}/\sqrt[3]{\log n}$, $m > b^{2/3}\sqrt[3]{\log n}$, and $k \geq n^2/m$.

Table 6.3: Dual objective separation for graphs.

	upper bound	lower bound	
Reach Disj.	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$	Safety Disj.
Büchi Disj.	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$	co-Büchi Disj.
Büchi Conj.	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$	co-Büchi Disj.
Streett	$\boldsymbol{n^2 + nk \log n}$	$\boldsymbol{k \cdot n^{2-o(1)}}$	Rabin

Significance of our results. We now describe the model and objective separation results that are obtained from the results we established.

(1) *Model Separation.* Table 6.2 shows our results that separate graphs and MDPs regarding their complexity for certain objectives and queries under assumptions (A1) and (A2). First, for reachability and Büchi objectives disjunction for graphs is in linear time while for MDPs we have $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for disjunctive queries. Second, for co-Büchi we have a separation when restricted to the class where each target set is a singleton. For these objectives disjunction on graphs is in linear time while we establish an $\Omega(m^{2-o(1)})$ conditional lower bound for MDPs for both disjunctive objectives and queries.

(2) *Objective Separation.* We next identify running time separations between different objectives. Here we consider two aspects, separations between dual objectives like Büchi and co-Büchi (Tables 6.3 and 6.4), and separations between conjunction and disjunction of objectives (Table 6.5). We compare dual objectives in two ways: (i) we show that single objectives that are dual to each other behave differently when we consider disjunction for each of them and (ii) we compare conjunctive objectives and their dual disjunctive objectives. For (ii) we have that conjunctive Büchi objectives are dual to disjunctive co-Büchi objectives, and Streett objectives, the conjunction of 1-pair Streett objectives, are dual to Rabin objectives, the disjunction of 1-pair Rabin objectives.

   (1) *Separating Dual Objectives for Graphs.* First, we consider reachability and safety objectives. For graphs we have that for reachability objectives disjunction is in linear time while for disjunctive safety objectives we establish an $\Omega(kn^{2-o(1)})$ lower bound under assumption (A1). Analogously, we have that disjunctive Büchi objectives are in linear time on graphs while we establish an $\Omega(kn^{2-o(1)})$ conditional lower bound for the disjunction of co-Büchi objectives. Furthermore, conjunctive Büchi objectives are in linear time and thus can be separated from their dual objective, the disjunctive co-Büchi objectives. Finally, for Streett objectives on graphs with $b = O(n^2/\log n)$ we have an $O(n^2)$ algorithm

Table 6.4: Dual objective separation for MDPs.

	upper bound	lower bound	
Büchi Disj. Obj.	$\min(n^2, m^{1.5}) + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	co-Büchi Disj. Obj.
Büchi Conj.	$\min(n^2, m^{1.5}) + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	co-Büchi Disj. Obj.
Streett	$\boldsymbol{\min(n^2, m^{1.5}\sqrt{\log n})}$ $\boldsymbol{+ nk \log n}$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$	Rabin

while we establish an $\Omega(n^{3-o(1)})$ lower bound for Rabin objectives when $k = \Theta(n)$. Note that $b \leq 2nk$.

(2) *Separating Dual Objectives for MDPs.* First, consider Büchi and co-Büchi objectives for MDPs. For MDPs disjunctive Büchi objectives are in time $O(\textsc{mec} + b)$, which is in $O(\min(n^2, m^{1.5}) + nk)$, while for co-Büchi objectives we show $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for both disjunctive queries and disjunctive objectives. This separates the two objectives for both sparse and dense graphs. Furthermore, conjunctive Büchi objectives can be solved in $O(\textsc{mec} + b)$ time and thus there is also a separation between disjunctive co-Büchi objectives and their dual. Finally, for MDPs with Streett objectives with $b = O(\min(n^2, m^{1.5}\sqrt{\log n})/\log n)$, we show both an $O(n^2)$ time and an $O(m^{1.5}\sqrt{\log n})$ time algorithm while we establish $\Omega(n^{3-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for Rabin objectives when $k = \Theta(n)$.

(3) *Separating Conjunction and Disjunction for Graphs and MDPs.* Except for reachability, i.e., in particular for all considered polynomial-time problems, we observe that the disjunction of objectives is computationally harder than the conjunction of these objectives (under assumptions (A1), (A2)). First, for safety objectives conjunction is in linear time even for MDPs while for disjunctive queries (disjunctive objectives are $PSPACE$-complete) we present $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds, where the first bound also holds for graphs. Second, for Büchi and co-Büchi objectives conjunction is in $O(\textsc{mec} + b)$ for MDPs (and $O(m+b)$ for graphs) while we show $\Omega(kn^{2-o(1)})$ and $\Omega(m^{2-o(1)})$ conditional lower bounds for disjunctive co-Büchi objectives and disjunctive Büchi / co-Büchi queries on MDPs. The $\Omega(m^{2-o(1)})$ bound even holds for the disjunction of singleton co-Büchi objectives. Furthermore, for co-Büchi objectives our $\Omega(kn^{2-o(1)})$ bound also holds on graphs, which separates conjunction and disjunction also in this setting. Third, we can also see the results for Streett and Rabin objectives as a separation between conjunction and disjunction. Recall that Streett objectives are

Table 6.5: Separating conjunction and disjunction.

		Conjunction	Disjunction
Safety	Graphs	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$
	MDP Qu.	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
Büchi	MDPs Qu.	$\min(n^2, m^{1.5}) + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
co-Büchi	Graphs	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\min(n^2, m^{1.5}) + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
1-pair Streett	Graphs	$\boldsymbol{n^2 + nk \log n}$	$\boldsymbol{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\boldsymbol{\min(n^2, m^{1.5}\sqrt{\log n}) + nk \log n}$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$
1-pair Rabin	Graphs	$m + nk$	$\boldsymbol{k \cdot n^{2-o(1)}}$
	MDPs Obj./Qu.	$\min(n^2, m^{1.5}) + nk$	$\boldsymbol{k \cdot n^{2-o(1)}, m^{2-o(1)}}$

the conjunction of one-pair Streett objectives and Rabin objectives are the disjunction of one-pair Rabin objectives. Further, both Büchi and co-Büchi objectives are special cases of each of one-pair Streett and one-pair Rabin objectives. In particular the following separations are easy observations or corollaries of our results: For the disjunction of one-pair Streett objectives the same conditional lower bounds (and the same upper bound, see Observation 6.5.11) as for the disjunction of co-Büchi objectives apply. Thus the disjunction of one-pair Streett objectives is harder than the conjunction of one-pair Streett objectives (under assumptions (A1)/(A2)). The conjunction of one-pair Rabin objectives can be solved in the same time as conjunctive Büchi objectives. Thus also the disjunction of one-pair Rabin objectives is harder than their conjunction.

Remark on Streett and Rabin objective separation. One remarkable aspect of our objective separation result is that we achieve it for Rabin and Streett objectives (both for graphs and MDPs), which are dual. For game graphs, Rabin objectives are NP-complete and Streett objectives are $coNP$-complete [EJ99]. For graphs and MDPs, both Rabin and Streett objectives can be solved in polynomial time. Since Rabin and Streett objectives are dual, and they belong to complementary complexity classes (either both in P, or one is NP-complete, other $coNP$-complete), they were considered to be equivalent for algorithmic purposes for graphs and MDPs. Quite surprisingly, we show that under some widely believed assumptions, both for MDPs and graphs, Rabin objectives are algorithmically harder than Streett objectives.

Remark on separating conjunction and disjunction. In logic disjunction and conjunction are dual and for polynomial-time problems, to the best of our knowledge, there have not been any results which show that one is harder than the other. For reachability objectives the conjunctive problems are harder (NP-complete for graphs, PSPACE-complete for MDPs) compared to the disjunctive problems, which are in polynomial time. In terms of strategy complexity, again conjunctive objectives are harder than disjunctive objectives: While for disjunctive Büchi objectives memoryless strategies suffice, for conjunctive Büchi objectives strategies require memory; and while for Rabin objectives memoryless strategies suffice, for Streett objectives strategies require memory (even exponential memory in games). Given the existing results, there was no evidence to expect that when polynomial-time algorithms exist that disjunction is harder than conjunction. On the contrary, existing results show that some aspects of conjunctive objectives are harder than for their disjunctive counterpart. Surprisingly, our results indicate that from an algorithmic point of view several polynomial-time problems are harder for the disjunctive problems than for their conjunctive counterparts.

Remark on digraph parameters. The graphs in the reductions for our lower bounds can be made acyclic by deleting a single vertex, thus our lower bounds also apply to a broad range of digraph parameters. For instance, let w be the DAG-width [Ber+06] of a graph, then we have for the problems with conditional lower bounds from (A1) that there is no $O(f(w) \cdot (k \cdot n^2)^{1-o(1)})$ time algorithm and for those with conditional lower bounds from (A2) that there is no $O(f(w) \cdot (km)^{1-o(1)})$ time algorithm under the respective assumptions.

Technical contributions. *Algorithms.* (1) We show that given the MECs of an MDP, the almost-sure winning set for a reachability objective can be determined in linear time on the MDP where each MEC is contracted to a player 1 vertex. This yields to the improved algorithms for disjunctive queries of reachability and Büchi objectives on MDPs. (2) For *MDPs* with *disjunctive co-Büchi* objectives and disjunctive queries of co-Büchi objectives we use the MECs in a different way; namely, we show that it is sufficient to do a linear-time computation within each MEC per co-Büchi objective to solve both disjunctive questions. (3) Finally we show that for *graphs* with a *disjunctive co-Büchi* objective for which the target set of each of the single co-Büchi objectives has *cardinality one*, the problem can be solved with a breadth-first search like algorithm in linear time.

Conditional Lower Bounds. (a) Conjecture (A1) is equivalent to the conjecture that there is no combinatorial $O(n^{3-\varepsilon})$ time algorithm to detect whether an n-vertex graph contains a triangle [VW10]. We show that triangle-detection in graphs can be linear-time reduced to *disjunctive queries of reachability objectives* on MDPs and thus that the running time for the latter can only be improved by sub-polynomial factors assuming (A1). (b) For the conditional lower bound under (A2) we consider the orthogonal vectors problem, which is known to be hard under (A2) [Wil05], and linear-time reduce it to *disjunctive queries of reachability objectives* on MDPs. (c) For *disjunctive safety problems* we give a linear-time reduction from triangle-detection that only requires player 1 vertices; thus the resulting conditional lower bound under (A1) also holds for graphs. (d) The reduction we give from the orthogonal vectors problem to *disjunctive safety problems* requires random vertices and thus the conditional lower bound under (A2) only holds for MDPs. (e) Based on the conditional lower bounds for disjunctive reachability queries and disjunctive safety problems, we then exploit reductions between the different types of objectives to obtain the conditional lower bounds for Büchi, co-Büchi, and Rabin.

6.2 Preliminiaries

Basic definitions and formal statements of the conjectures can be found in Chapter 2. Given c objectives $\phi_1, \dots, \phi_c$, the *conjunctive objective* $\phi = \phi_1 \cap \dots \cap \phi_c = \bigwedge_{i=1}^{c} \phi_i$ is given by the intersection of the c objectives, and the *disjunctive objective* $\phi = \phi_1 \cup \dots \cup \phi_c = \bigvee_{i=1}^{c} \phi_i$ is given by the union of the c objectives. For the *conjunctive query* of c objectives $\phi_1, \dots, \phi_c$ we define the (almost-sure) winning set to be the set of vertices that have one strategy that is (almost-sure) winning for *each of* the objectives $\phi_1, \dots, \phi_c$. Analogously, a vertex is in the (almost-sure) winning set $\bigvee_{i=1}^{c} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \phi_i\right)$ for the *disjunctive query* of the c objectives if it is in an (almost-sure) winning set for *at least one* of the c objectives (i.e. we take the union of the winning sets).

Below we present several observations that interlink different types of objectives.

Observation 6.2.1. *The almost-sure winning set for a* conjunctive objective *is the same as for the corresponding* conjunctive query.

Proof. We have for any $v \in V$ and $\sigma \in \Sigma$ and any two objectives ϕ_1, ϕ_2 that $\Pr_v^\sigma \left(\phi_1 \wedge \phi_2\right) = 1$ iff $\Pr_v^\sigma \left(\phi_1\right) = 1$ and $\Pr_v^\sigma \left(\phi_2\right) = 1$. □

Observation 6.2.2. *On* graphs (*i.e.* $V_R = \emptyset$) *the winning set for a* disjunctive objective *is the same as for the corresponding* disjunctive query.

Proof. For any two objectives ϕ_1, ϕ_2 we have for each $\omega \in \Omega$ that $\omega \in (\phi_1 \cup \phi_2)$ iff $\omega \in \phi_1$ or $\omega \in \phi_2$. □

Observation 6.2.3. *The* disjunctive objective *of* Büchi (*resp. reachability*) *objectives is the same as the Büchi* (*resp. reachability*) *objective of the union of the target sets.*

Proof. We show the claim for Büchi, the proof for reachability is analogous. For two target sets $T_1, T_2 \subseteq V$ we have $\{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_1 \neq \emptyset\} \cup \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap T_2 \neq \emptyset\} = \{\omega \in \Omega \mid \mathrm{Inf}(\omega) \cap (T_1 \cup T_2) \neq \emptyset\}$. □

Observation 6.2.4. *The* conjunctive objective *of* co-Büchi (*resp. safety*) *objectives is the same as the co-Büchi* (*resp. safety*) *objective of the union of the target sets.*

Proof. We show the claim for co-Büchi, the proof for safety is analogous. For two target sets $T_1, T_2 \subseteq V$ we have $\{\omega \in \Omega \mid \text{Inf}(\omega) \cap T_1 = \emptyset\} \cap \{\omega \in \Omega \mid \text{Inf}(\omega) \cap T_2 = \emptyset\} = \{\omega \in \Omega \mid \text{Inf}(\omega) \cap (T_1 \cup T_2) = \emptyset\}$. □

By definition each path winning for a safety objective is also winning for the corresponding co-Büchi objective while the converse is not always true. However, when it comes to the non-emptiness of winning sets, these two objectives become equivalent.

Observation 6.2.5. *For a fixed MDP P the winning set for Safe(T) is non-empty iff the winning set for coBüchi(T) is non-empty. This equivalence extends also to conjunctions and disjunctions of safety and co-Büchi objectives.*

Proof. By [CY95, p. 891] (see also Section 2.6.2) the winning set for Safe(T) resp. coBüchi(T) is non-empty if and only if there exists an end-component X with $X \cap T = \emptyset$. □

Observation 6.2.6. *Disjunctive reachability queries (resp. objectives) in MDPs can be linear time reduced to disjunctive Büchi queries (resp. objectives) in MDPs and vice versa.*

Proof. Reachability ⇒ Büchi: For each target set T replace each $t \in T$ with two vertices: $t_{\text{in}} \in V_1$ and t_{out}, where t_{out} belongs to the same player as t. Assign all incoming edges of t to t_{in} and all outgoing edges of t to t_{out}, and add the edge $(t_{\text{in}}, t_{\text{out}})$ and the self-loop $(t_{\text{in}}, t_{\text{in}})$. Let the corresponding target set T' for Büchi consist of the vertices t_{in} for all $t \in T$. As player 1 can force the play to stay at a vertex t_{in} as soon as it is reached, we have that the set T' in the modified MDP can be visited infinitely often almost surely iff in the original MDP the set T can be reached almost surely.

Büchi ⇒ reachability: For each target set T replace each $t \in T$ with three vertices: $t_{\text{in}} \in V_R$, $t_r \in V_1$, and t_{out}, where t_{out} belongs to the same player as t. Assign all incoming edges of t to t_{in} and all outgoing edges of t to t_{out}, and add the edges $(t_{\text{in}}, t_{\text{out}})$, (t_{in}, t_r), and (t_r, t_{out}). Let the corresponding target set T' for reachability consist of the vertices t_r for all $t \in T$. Notice that the vertex t_r can only be reached from t_{in} and that whenever t_{in} is reached, the vertex t_r is reached with probability 1/2. Furthermore, the vertex t_{out} is reached whenever the vertex t_{in} is reached. Notice that there is a one-to-one correspondence between player-1 strategies for the two MDPs. Thus when player 1 follows the corresponding strategies for both MDPs, then the set T' in the modified MDPs is a.s. reached if and only if the set T is a.s. reached infinitely often. Hence the correctness of the reduction follows. □

Observation 6.2.7. *Conjunctive Büchi* (*resp. co-Büchi*) *objectives are special instances of Streett objectives.*

Proof. For Büchi let $L_j = V$ and $U_j = T_j$, for co-Büchi let $L_j = T_j$ and $U_j = \emptyset$. $\square$

Observation 6.2.8. *Disjunctive Büchi (resp. co-Büchi) objectives are special instances of Rabin objectives.*

Proof. For Büchi let $L_j = T_j$ and $U_j = \emptyset$, for co-Büchi let $L_j = V$ and $U_j = T_j$. $\square$

6.3 Reachability in MDPs

First let us briefly discuss reachability in graphs. The winning set for disjunctive reachability can simply be computed by taking the union of all target sets and then starting a breadth-first search, which is in $O(m)$. On the other hand, the problem becomes NP-complete when considering conjunctive reachability [FH10], as with conjunction one can require a path to contain several vertices and in particular one can embed the well-known NP-hard problem of Hamiltonian path.

Turning to MDPs, notice that in MDPs based on acyclic graphs almost-sure reachability is equivalent to computing the winning set for a player with reachability objectives in a game graph where all the random vertices are owned by the opponent (as the random player plays the optimal strategy for the opponent with non-zero probability). As computing the winning set in a conjunctive reachability game is $PSPACE$-hard even for acyclic graphs [FH10], we have that conjunctive almost-sure reachability in MDPs is $PSPACE$-hard as well.

In the first part of this section we present an improved algorithm for disjunctive reachability *queries* in MDPs. As a disjunctive reachability *objective* can easily be reduced to a single reachability objective by taking the union of all target sets, this algorithm with $k = 1$ is also an algorithm for disjunctive reachability objectives (but does not improve upon the known running time for this case). We refer to computing the almost-sure winning set for a single reachability objective as the *almost-sure reachability* problem. In the second part we present two lower bounds for disjunctive reachability queries, an $\Omega(n^{3-o(1)})$ lower bound based on CTC (resp. BMM) and an $\Omega(m^{2-o(1)})$ lower bound based on OVC (resp. SETH).

6.3.1 Algorithm for Disjunctive Reachability Queries in MDPs

In this section we present an algorithm to compute the almost-sure winning set for disjunctive reachability queries in MDPs. In particular we show the following theorem, where the result for Büchi objectives follows from the result for reachability objectives by Observation 6.2.6.

Theorem 6.3.1. *For an MDP P and target sets $T_j \subseteq V$ for $1 \leq j \leq k$ the almost-sure winning set for the disjunctive query of Reach(T_j) for $1 \leq j \leq k$ can be computed in $O(km + \text{MEC})$ time. The same bound holds for the disjunctive query of Büchi(T_j).*

A vertex v is in the almost-sure winning set iff player 1 has a strategy to reach one of the k target sets T_j with probability 1 starting from v. Note that the sets T_j

are not necessarily absorbing in contrast to what is often assumed for the reachability objective in MDPs. The trivial algorithm would be to invoke an algorithm for almost-sure reachability in MDPs k times (for one target set T_j at a time, temporarily making the set T_j absorbing if necessary). The crucial observation to improve upon this is that given an MDP without non-trivial end-components, almost-sure reachability in MDPs can be solved in linear time. Recall from Section 2.5.2 that a non-trivial end-component is an end-component with at least two vertices. We further observe that, for each target set, either all vertices of an end-component are a.s. winning or none. Thus if we know the MEC-decomposition of an MDP, we can contract the MECs to single vertices with self-loops and solve almost-sure reachability on the derived MDP. This derived MDP does not have non-trivial end-components, therefore, given the MEC-decomposition, the problem can be solved in linear time per target set. Our algorithm implies that almost-sure reachability (i.e. the case $k = 1$) can be solved in the same asymptotic running time as computing the MEC-decomposition of an MDP; thus any improvements over the known $O(\min(n^2, m^{1.5}))$ running time for MEC-decomposition [CH14] carry over to almost-sure reachability. The same running time is achieved for almost-sure reachability in [CJH03; CH14], i.e., our algorithm improves upon the known running times only for super-constant k.

Definition 6.3.2 (Contraction of MECs). *Contracting a MEC X in an MDP P creates a modified MDP P' from P where the vertices of X are replaced by a single vertex u that belongs to player 1 and the edges to or from a vertex in X are replaced with edges to or from, respectively, the vertex u; parallel edges are omitted from P', for parallel random edges the probabilities are added up.*

Observation 6.3.3 ([CH11]). *The MDP P' that is constructed from the MDP P by contracting all MECs of P does not contain any non-trivial end-components.*

Proof. Assume by contradiction that the MDP P' contains an end-component X' with at least two vertices. Let X be the set of vertices corresponding to the vertices of X' in the original MDP P. Then X is an end-component in P, a contradiction to the definition of P'. □

In the derived MDP we apply, for each target set, one iteration of the classical almost-sure reachability algorithm but with a modified random attractor computation defined below. The classical algorithm repeatedly executes the following two steps: 1) Compute the vertices S from which player 1 can reach the target set T. 2a) If $S = V$, output S as the a.s. winning set of player 1. 2b) If $S \subsetneq V$, remove the random attractor of $V \setminus S$ from the graph and repeat. Recall from Section 2.5.2 that a *random attractor* of a set of vertices W contains the vertices from which there is a positive probability to reach W for every strategy of player 1. The *extended random attractor*, formally defined below and used implicitly in [CH14], additionally includes player 1 vertices for which the only player 1 strategy to avoid a positive probability to reach W is using a self-loop of a vertex not in the target set. Additionally, we explicitly avoid adding vertices in the considered target set to the attractor. In the classical algorithm this was achieved by making the target set absorbing.

Definition 6.3.4 (Extended Random Attractor). *In an MDP $P = ((V, E), (V_1, V_R), \delta)$ the* extended random attractor *$Attr^+_R(P, W, T)$ for sets of vertices $W, T \subseteq V$ is defined as $Attr^+_R(P, W, T) = \bigcup_{i \geq 0} Z_i$ where $Z_0 = W \setminus T$ and Z_{i+1} for $i \geq 0$ is defined as $Z_{i+1} = Z_i \cup \{v \in V_R \mid Out(v) \cap Z_i \neq \emptyset\} \cup \{v \in V_1 \mid Out(v) \subseteq (Z_i \cup \{v\})\} \setminus T$.*

In contrast to a random attractor (a) a set of vertices T can be specified that is never included in $Attr^+_R(P, W, T)$ and (b) a player 1 vertex is also included in Z_{i+1} if all its outgoing edges apart from its self-loop are contained in Z_i. An extended random attractor $A = Attr^+_R(P, W, T)$ can be computed in $O(\sum_{v \in A} Indeg(v) + |V_1 \setminus T|)$ time [Bee80; Imm81].

The overall algorithm for MDPs with disjunctive reachability queries is described in Algorithm DisjReachMDP. First, the MEC-decomposition of the input MDP P is computed (l. 1). Then all MECs of P are contracted to construct the derived MDP P', which does not contain any non-trivial MECs (l. 2). For each target set we execute one iteration of the classical algorithm (l. 5–8), replacing the usual random attractor with the extended random attractor (l. 7). After reversing the contraction (l. 9), the union of the winning sets determined for each target set yields the a.s. winning set of player 1 for the disjunctive reachability query.

Algorithm DisjReachMDP: MDPs with disjunctive reachability queries

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target sets $T_j \subseteq V$ for $1 \leq j \leq k$
Output : $\bigvee_{1 \leq j \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_j))$
1  compute MEC decomposition of $P$
2  let $P'$ be $P$ with all MECs contracted
3  let $T'_j$ for $1 \leq j \leq k$ be the set of vertices of $P'$ that represent some vertex of $T_j$
4  $W' \leftarrow \emptyset$
5  for $j \leftarrow 1$ to $k$ do
6      $S' \leftarrow$ GraphReach$(P', T'_j)$
7      $A' \leftarrow Attr^+_R(P', V' \setminus S', T'_j)$
8      $W' \leftarrow W' \cup (V' \setminus A')$
9  let $W$ be the vertex set resulting from $W'$ by reversing the contraction
10 return $W$
```

Proposition 6.3.5 (Running time). *Algorithm DisjReachMDP runs in time $O(km +$ MEC$)$.*

Proof. By definition the computation of all MECs takes $O($MEC$)$. Contracting all MECs can be done in time $O(m)$ as we have to consider each edge (and vertex) at most twice. The for-loop is executed k times. Within the for-loop, both the vertices S that can reach T_j and the extended random attractor $A = Attr_R^+(P', V' \setminus S', T_j')$ can be found in linear time, that is, in $O(km)$ time over all iterations of the for-loop. Undoing the contraction takes again at most $O(m)$ time. $\square$

Towards proving the correctness of Algorithm DisjReachMDP, first note that by definition the disjunctive query is satisfied almost-surely for a vertex if and only if the reachability objective is satisfied almost-surely for one of the target sets. Hence we can consider the k target sets separately by showing that in the j-th iteration of the for-loop of Algorithm DisjReachMDP the set $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \text{Reach}(T_j)\right)$ is identified.

In the correctness proof we assume that in the MDP P each vertex has at least one outgoing edge and each random vertex has at least one outgoing edge that is not a self-loop. This is w.l.o.g. because $\bigvee_{1 \le j \le k} \langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \text{Reach}(T_j)\right)$ does not change if we replace each vertex without outgoing edges by a vertex with a self-loop and treat a random vertex whose only outgoing edge is a self-loop as a player 1 vertex.

Notation. Let P' be the MDP derived from the MDP P by contracting all MECs of P and let T_j' be the set of contracted vertices that represent some vertex of T_j as in Algorithm DisjReachMDP. We use the superscript $'$ to denote sets related to the MDP P' and omit the superscript for sets related to the original MDP P. Note that since only strongly connected subgraphs are contracted in P', it clearly holds that a vertex $v \in V$ can reach another vertex $u \in V$ if and only if the vertex $v' \in V'$ into which v is contracted to can reach the vertex $u' \in V'$ into which u is contracted to.

Fix some iteration j and let $S' = \text{GraphReach}(P', T_j')$, let $A' = Attr_R^+(P', V' \setminus S', T_j')$, and let $W_j' = V' \setminus A'$, that is, W_j' is the set added to W' in the j-th iteration of the for-loop of Algorithm DisjReachMDP. Let the same letters without superscript denote the corresponding sets of vertices after reverting the contraction of the MECs of P. We prove the correctness of the algorithm by first showing $W_j \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \text{Reach}(T_j)\right)$ and then $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \text{Reach}(T_j)\right) \subseteq W_j$.

Lemma 6.3.6. *For each $1 \le j \le k$ we have $W_j \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(T_j)\right)$.*

Proof. Let $G[W_j] = (W_j, E \cap (W_j \times W_j))$ be the subgraph induced by the vertices of W_j. We establish two properties: (1) all outgoing edges of random vertices $V_R \cap W_j$ lead to vertices in W_j, and (2) all vertices in $W_j \setminus T_j$ can reach T_j in $G[W_j]$. The claim follows from these two properties using the same proof as for the classical algorithm for almost-sure reachability in MDPs (folklore, see below).

(1) For vertices in V_R we distinguish whether they are contained in a MEC of P or not. In the first case, property (1) follows from the fact that a MEC has no outgoing random edges and every MEC is either completely contained in W_j or completely contained in $V \setminus W_j$. In the second case, property (1) follows from the definition of an extended random extractor because a vertex of $V_R \cap W_j'$ with an edge to a vertex of $A = V \setminus W_j$ would have been included in the (extended) random attractor A'.

(2) To show that every vertex in $W_j \setminus T_j$ can reach T_j in $G[W_j]$, we use that by Observation 6.3.3 the MDP P' does not contain any non-trivial MEC. Assume by contradiction that some vertices in $W_j \setminus T_j$ cannot reach T_j in $G[W_j]$. Then there exists a bottom SCC C (i.e. an SCC without outgoing edges, possibly a single vertex) in $G[W_j]$ with $C \cap T_j = \emptyset$. Note that every MEC in $G[W_j]$ is completely contained in one of the SCCs of $G[W_j]$. By property (1) C has no outgoing random edges in P; by this and the fact that C is strongly connected, the corresponding set C' of contracted vertices in P' would be a non-trivial MEC in P' if it contained more than one vertex. Thus C' can contain only one vertex c' and this vertex has either no outgoing edge or only a self-loop in $G'_{W'_j}$. If c' was a player 1 vertex, then all its outgoing edges would go to vertices of $A' = V' \setminus W'_j$ or be a self-loop, hence c' would have been included in the extended random attractor A'. If c' was a random vertex, then by the assumption that in P, and thus in P', every random vertex has an outgoing edge that is not a self-loop we would get a contradiction to property (1). Thus no such bottom SCC C can exist, that is, every bottom SCC of $G[W_j]$ contains a vertex of T_j and thus property (2) holds.

To see that the two established properties imply $W_j \subseteq \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Reach}(T_j)\right)$, let for a vertex $u \in W_j$ be $d(u)$ the shortest path distance to a vertex in T_j. Consider the following strategy σ of player 1: For a player 1 vertex u, choose an edge to a vertex v such that $d(v) < d(u)$. For a random vertex u, there is always an edge to a vertex v such that $d(v) < d(u)$. Let $\ell = |W_j|$ and let α be the minimum positive transition probability in the MDP P. For all vertices $v \in W_j$ the probability that T_j is reached within ℓ steps is at least α^{ℓ}, that is, the probability that T_j is not reached within $b \cdot \ell$ steps is at most $(1 - \alpha^{\ell})^b$, which goes to 0 as b goes to ∞. Thus for all $v \in W_j$ strategy σ ensures that T_j is reached with probability 1. □

Lemma 6.3.7. *For each* $1 \leq j \leq k$ *we have* $\langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \mathit{Reach}(T_j)\right) \subseteq W_j$.

Proof. We prove $\langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Reach}(T_j)\right) \subseteq W_j$ by showing that $A = V \setminus W_j$ is contained in $V \setminus \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Reach}(T_j)\right)$, that is, that we have for all vertices $a \in A$ and every strategy σ of player 1 that $\Pr_a^{\sigma} \left(P, \text{Reach}(T_j)\right) < 1$. We show this by induction on the recursive definition of $A' = \mathit{Attr}_R^{+}(P', V' \setminus S', T'_j) = \bigcup_{i \geq 0} Z'_i$, where the sets Z'_i are defined as in Definition 6.3.4 and the sets Z_i are the corresponding sets after reverting the contraction of the MECs of P. Since the attractor computation is done on P', each set Z_i either contains all vertices of a MEC of P or none. Clearly $A \cap T_j = \emptyset$ as vertices in T'_j are explicitly excluded from A'. Player 1 cannot reach T_j almost surely from the vertices in $Z_0 = V \setminus S$ because these vertices cannot reach any vertex in T_j. Assume the claim holds for Z_i, i.e., for all vertices $z \in Z_i$ and any strategy σ of player 1 we have $\Pr_z^{\sigma} \left(P, \text{Reach}(T_j)\right) < 1$. By the definition of Z'_{i+1}, for a random vertex v' in $Z'_{i+1} \setminus Z'_i$ there is a positive probability to reach a vertex in Z'_i; thus, $\Pr_{v'}^{\sigma'} \left(P', \text{Reach}(T'_j)\right) < 1$ for any strategy σ' of player 1.

Random vertices in P' are not contracted, thus the same argument holds for Z_{i+1} and P. A player-1 vertex x' in $Z'_{i+1} \setminus Z'_i$ corresponds to either a player-1 vertex x or a MEC X in $Z_{i+1} \setminus Z_i$. In both cases all the edges from x resp. X lead to vertices in Z_i or to x resp. X itself. Hence since $x \notin T_j$ resp. $X \cap T_j = \emptyset$, we also have $\Pr_x^\sigma(\text{Reach}(T_j)) < 1$ for any strategy σ of player 1 and x resp. all $x \in X$. □

Proposition 6.3.8 (Correctness). *For an MDP P and target sets $T_j \subseteq V$ for $1 \le j \le k$ Algorithm DisjReachMDP returns the set $\bigvee_{1\le j\le k} \langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathit{Reach}(T_j))$.*

Proof. We have that a vertex is in $\bigvee_{1\le j\le k} \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j))$ if and only if it is in $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j))$ for some $1 \le j \le k$. Thus it is sufficient to show that in the j-th iteration of the for-loop of Algorithm DisjReachMDP the set $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j))$ is identified. Let W'_j denote the set of vertices in the contracted MDP P' that is identified in iteration j and let W_j denote the set of vertices obtained from W'_j by reverting the contraction of the MECs. By Lemma 6.3.6 we have $W_j \subseteq \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j))$ and by Lemma 6.3.7 we have $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j)) \subseteq W_j$. Hence the union of the sets W_j that is returned by the algorithm is equal to $\bigvee_{1\le j\le k} \langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Reach}(T_j))$. □

6.3.2 Conditional Lower Bounds for Disjunctive Reachability in MDPs

Here we complement the above algorithm by conditional lower bounds for disjunctive reachability queries in MDPs. These lower bounds are based on the conjectures CTC and OVC introduced in Section 2.7. We first present our lower bound on the worst-case running time in terms of k and n based on CTC (which is equivalent to BMM) that is particularly relevant for dense graphs with $m = \Theta(n^2)$.

Theorem 6.3.9. *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for disjunctive reachability queries in MDPs under Conjecture 2.7.3, i.e., unless CTC and BMM fail. In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

We show the theorem using the following reduction from triangle detection.

Reduction 6.3.10. *Given an instance of the triangle detection problem with a graph $G = (V, E)$, we build the following MDP $P = ((V', E'), (V'_1, V'_R))$. The target sets for the disjunctive reachability queries on P are given by $T_v = \{g_v\}$ for g_v as defined below.*

- *The vertices V' of P are given by four copies V^1, V^2, V^3, V^4 of V, a start vertex s, and* absorbing *vertices $F = \{g_v \mid v \in V\}$. The edges E' of P are defined as follows: For $1 \le i \le 3$ there is an edge from v^i to u^{i+1} iff $(v, u) \in E$; furthermore, there is an edge from s to the first copy $v^1 \in V^1$ of every $v \in V$, and*

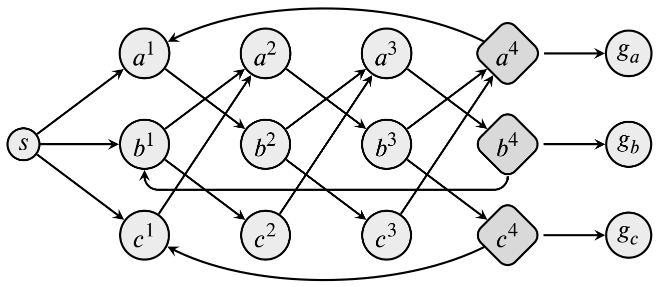

Figure 6.1: Illustration of Reduction 6.3.10, with $G = (\{a, b, c\}, \{(a, b), (b, a), (b, c), (c, a)\})$. Cycles denote player-1 vertices, diamonds random vertices.

the last copy $v^4 \in V^4$ of every $v \in V$ is connected to its first copy v^1 and its corresponding absorbing vertex $g_v \in F$; each absorbing vertex has a self-loop.

- *The set of vertices V' is partitioned into player-1 vertices $V_1' = \{s\} \cup V^1 \cup V^2 \cup V^3 \cup F$ and random vertices $V_R' = V^4$. Moreover, the probabilistic transition function for each vertex $v \in V_R'$ chooses among v's successors uniformly at random, i.e., with probability 1/2 each.*

The reduction is illustrated in Figure 6.1. As all random choices are uniformly at random, we omit the exact probabilities in the figures.

Next we prove that Reduction 6.3.10 is indeed a valid reduction from triangle detection to disjunctive reachability queries in MDPs.

Lemma 6.3.11. *Let P be the MDP and T_v for $v \in V$ the target sets given by Reduction 6.3.10 for a graph $G = (V, E)$. The graph G contains a triangle iff s is contained in $\bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as} (P, Reach(T_v))$.*

Proof. For the only if part assume that G has a triangle with vertices a, b, c and let a^i, b^i, c^i be the copies of a, b, c in V^i. Now a strategy for player 1 in the MDP P to reach g_a with probability 1 is as follows: When at s, go to a^1; when at a^1, go to b^2; when at b^2, go to c^3; when at c^3, go to a^4. As a, b, c form a triangle, all the edges required by the above strategy exist. When player 1 starts at s and follows the above strategy, the only random vertex she encounters is a^4. The random choice sends her to the target vertex g_a and to vertex a^1 with probability 1/2 each. In the former case, she is done, in the latter case she continues playing her strategy and reaches a^4 again after three steps. The probability that player 1 reaches g_a in at most $3q + 1$ steps is $1 - (1/2)^q$ which converges to 1 when q goes to infinity. Thus we have found a strategy to reach g_a with probability 1.

For the if part assume that $s \in \bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_v))$. That is, there is an $a \in V$ such that $s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_a))$. Let us consider a corresponding strategy for reaching $T_a = \{g_a\}$. First, assume that a play resulting from the strategy would visit a vertex v^4 for $v \in V \setminus \{a\}$. Then with probability 1/2 the play would reach the vertex g_v, which has no path to g_a, a contradiction to $s \in \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Reach}(T_a)\right)$. Thus the strategy has to prohibit that a play visits vertices v^4 for $v \in V \setminus \{a\}$. Second, as the only way to reach g_a is a^4, the strategy has to choose a^4. But then with probability 1/2 the play goes to a^1 and there must be a path from a^1 to g_a that does not cross $V^4 \setminus \{a^4\}$. By the latter this path must be of the form a^1, b^2, c^3, a^4, g_a for some $b, c \in V$. By the construction of the MDP P, the vertices a, b, c form a triangle in the original graph G. □

The size and the construction time of the MDP P, constructed by Reduction 6.3.10, is linear in the size of the original graph G and we have $k = \Theta(n)$ target sets. Thus if we would have a combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm for disjunctive queries of reachability objectives in MDPs for some $\epsilon > 0$, we would immediately get a combinatorial $O(n^{3-\epsilon})$ time algorithm for triangle detection, which contradicts CTC and BMM.

Next we present a lower bound on the running time in terms of k and m based on OVC (and thus SETH) that is particularly relevant for sparse graphs.

Theorem 6.3.12. *There is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for disjunctive reachability queries in MDPs under Conjecture 2.7.5, i.e., unless OVC and SETH fail. In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

To prove the above, we give a reduction from the orthogonal vectors problem to disjunctive reachability queries in MDPs.

Reduction 6.3.13. *Given two sets S_1 and S_2 of d-dimensional vectors, we build the following MDP $P = ((V, E), (V_1, V_R))$. The target sets for the disjunctive reachability query are defined as $T_v = \{g_v\}$ for g_v as defined below.*

- *The vertices V of the MDP P are given by a start vertex s, vertices S_1 and S_2 representing the sets of vectors, vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates, and absorbing vertices $F = \{g_v \mid v \in S_2\}$. The edges E of P are defined as follows: for each $x \in S_1$ there is an edge* to *$c_i \in \mathcal{C}$ iff $x_i = 1$ and for each $y \in S_2$ there is an edge* from *$c_i \in \mathcal{C}$ iff $y_i = 0$; further, the start vertex s has an edge to every vertex of S_1 and every vertex $v \in S_2$ has an edge to s and to its corresponding absorbing vertex $g_v \in F$; every absorbing vertex has a self-loop.*

- *The set of vertices V is partitioned into player 1 vertices $V_1 = \{s\} \cup \mathcal{C} \cup F$ and random vertices $V_R = S_1 \cup S_2$. The probabilistic transition function for each vertex $v \in V_R$ chooses among v's successors uniformly at random.*

The reduction is illustrated on an example in Figure 6.2.

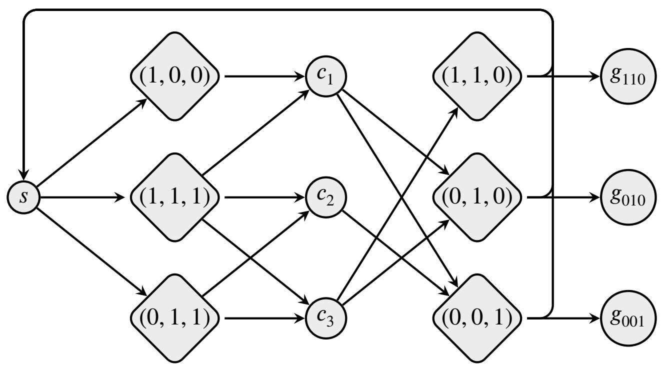

Figure 6.2: Illustration of Reduction 6.3.13 for $S_1 = \{(1, 0, 0), (1, 1, 1), (0, 1, 1)\}$ and $S_2 = \{(1, 1, 0), (0, 1, 0), (0, 0, 1)\}$.

Lemma 6.3.14. *Let $P = ((V, E), (V_1, V_R))$ be the MDP and $T_v = \{g_v\}$ for $v \in V$ the target sets given by Reduction 6.3.13 for an instance of the orthogonal vectors problem specified by S_1 and S_2. There exist orthogonal vectors $x \in S_1$ and $y \in S_2$ iff $s \in \bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as} (P, Reach(T_v))$.*

Proof. If one of the sets S_1 and S_2 contains the all-zero vector, an orthogonal pair of vectors exists trivially and this can be detected in linear time; thus we assume w.l.o.g. that none of the sets contains the all-zero vector. To ensure that each vertex in the constructed MDP contains an outgoing edge, we further assume that for each coordinate i there exists a vector $y \in S_2$ with $y_i = 0$. This can be ensured within the construction time of P by removing vertices c_i without outgoing edges and all vertices $x \in S_1$ with an edge to such a coordinate vertex. This does not change the orthogonal vector instance as every $y \in S_2$ has $y_i = 1$ if c_i does not have an outgoing edge and thus there cannot exist an orthogonal pair that contains a vector $x \in S_1$ with $x_i = 1$.

For the only if part assume that there are orthogonal vectors $x \in S_1$ and $y \in S_2$. Now a strategy for player 1 in the MDP P to reach g_y with probability 1 is as follows: When at s, go to x; when at some $c \in \mathscr{C}$, go to y. As x and y are orthogonal, each $c_i \in \mathscr{C}$ reachable from x has an edge to y, i.e., for $x_i = 1$ it must be that $y_i = 0$. When player 1 starts at s and follows the above strategy, she reaches y after three steps. There the random choice sends her, with probability 1/2 each, either to the target vertex g_y or back to vertex s. In the former case, she is done, in the latter case she continues playing her strategy and reaches y again after three steps. The probability that player 1 reaches g_y in at most $3q$ steps is $1 - (1/2)^q$, which converges to 1 when q goes to infinity. Thus this strategy reaches g_y with probability 1.

For the if part assume that $s \in \bigvee_{v \in V} \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_v))$. That is, there is an $y \in S_2$ such that $s \in \langle\!\langle 1 \rangle\!\rangle_{as} (P, \text{Reach}(T_y))$. Let us consider a corresponding strategy for reaching $T_y = \{g_y\}$. First, assume that a play resulting from the strategy would visit a vertex $y' \in S_2$ for $y' \neq y$. Then with probability 1/2 the player would end up in the vertex $g_{y'}$, which has no path to g_y, a contradiction to $s \in \langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \text{Reach}(T_y)\right)$. Thus the strategy has to prohibit that a play visits vertices of $S_2 \setminus \{y\}$. Second, as the only way to reach g_y is from y, the strategy has to choose y. But then with probability 1/2 the play goes to s and thus there must be a strategy to reach g_y from s with probability 1 that does not cross $S_2 \setminus \{y\}$. As y is the only predecessor of g_y, there must also be such a strategy to reach y. In other words, there must be an $x \in S_1$ such that for each successor $c_i \in \mathscr{C}$ there is an edge to y. By the construction of the MDP P this is equivalent to the existence of an $x \in S_1$ such that whenever $x_i = 1$ then $y_i = 0$, and thus x and y are orthogonal vectors. $\square$

The number of vertices in P, constructed by Reduction 6.3.13, is $O(N)$ and the construction can be performed in $O(N \log N)$ time (recall that $d \in O(\log N)$). The number of edges m is $O(N \log N)$ (thus we consider P to be a sparse MDP) and the number of target sets $k \in \Theta(N) = \theta(m/\log N)$. Finally, if we would have an $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm for disjunctive reachability queries in MDPs for any $\epsilon > 0$, we would immediately get an $O(N^{2-\epsilon})$ time algorithm for OV, which contradicts OVC (and thus SETH).

6.4 Safety Objectives

It is well-known that computing the almost-sure (a.s.) winning set for a single safety objective in an MDP is equivalent to computing the winning set of player 1 in game graphs with safety objectives, where all the random vertices are owned by player 2 (see e.g. [CDH10]). Safety objectives in game graphs can be computed in $O(m)$ time by computing a player-2 attractor to the target set and taking the complement (see Section 2.5.3). Thus in MDPs the a.s. winning set for a single safety objective can be computed in $O(m)$ time by computing a random attractor, and the a.s. winning set for a disjunctive query can be determined in $O(k \cdot m)$ time by computing k random attractors and taking the union of the winning sets. For safety objectives memoryless strategies suffices, see, e.g., [Tho95].

By Observation 6.2.4 conjunctive safety objectives (which coincide with conjunctive queries by Observation 6.2.1) can be reduced to single safety objectives in $O(b)$ time by taking the union of all the sets T_i.

Turning to disjunctive safety objectives, we have the same equivalence to game graphs as for single objectives (see Observation 6.4.1 below). In a game graph the disjunctive safety objective is the complementary objective to the conjunctive reachability objective with the same sets and, since the winning sets of the two players form a partition of the vertices of the game graph, the $PSPACE$-hardness shown in [FH10] also applies to disjunctive safety objectives.

Observation 6.4.1. *Computing the a.s. winning set for a disjunctive safety objective in an MDP with player-1 vertices V_1 and random vertices V_R is equivalent to computing the winning set in a game graph with the same disjunctive safety objective, where the* *game graph is obtained from the MDP by assigning the random vertices to player 2, i.e., $V_2 = V_R$.*

Proof. We show that a vertex s is almost-sure winning in the MDP if and only if it is winning for player 1 in the game graph.
$\Rightarrow$: Assume s is not winning for player 1 in the graph game. Then s is winning for player 2 and thus player 2 has a strategy to visit all target sets from s. As there are only finitely many target sets, all these target sets are visited after a finite number of steps, lets say after ℓ steps. Now consider the corresponding MDP; with some constant probability the random choices in the MDP follows exactly the strategy of player 2 in the graph game for the first ℓ steps and thus all target sets are reached. In this case player 1 cannot win almost surely from s. Hence, s is not in the a.s. winning set.
$\Leftarrow$: Assume player 1 has a winning strategy for the graph game starting in s. Since this strategy is winning for each possible choice of player 2, it also winning for a random choice. $\square$

6.4.1 Conditional Lower Bounds for Safety Objectives

We first present a lower bound for disjunctive safety objectives and queries based on CTC that even holds on graphs. Recall that on graphs disjunctive objectives and queries coincide.

Theorem 6.4.2. *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for disjunctive safety objectives (or queries) in graphs under Conjecture 2.7.3, i.e., unless CTC and BMM fail. In particular, there is no such algorithm for deciding whether the winning set is non-empty or for deciding whether a specific vertex is in the winning set.*

The above is by the following linear time reduction from triangle detection to disjunctive safety objectives/queries in graphs.

Reduction 6.4.3. *Given an instance of the triangle detection problem, i.e., a graph $G = (V, E)$, we build a graph $G' = (V', E')$ and disjunctive safety objectives as follows. The set of vertices V' is the union of four copies V^1, V^2, V^3, V^4 of V and a vertex s. For $1 \leq i \leq 3$ a vertex $v^i \in V^i$ has an edge to a vertex $u^{i+1} \in V^{i+1}$ iff $(v, u) \in E$. The vertex s has an edge to all vertices in V^1 and all vertices in V^4 have an edge to s. The target sets are given by $T_v = (V^1 \setminus \{v^1\}) \cup (V^4 \setminus \{v^4\})$ for $v \in V$.*

Reduction 6.4.3 is illustrated in Figure 6.3.

Lemma 6.4.4. *Let G' be the graph and T_v for $v \in V$ the target sets given by Reduction 6.4.3 for a graph $G = (V, E)$. The following statements are equivalent.*

(1) *G contains a triangle.*

(2) *The vertex s is in the winning set of $(G', \bigvee_{v \in V} \mathit{Safe}(T_v))$.*

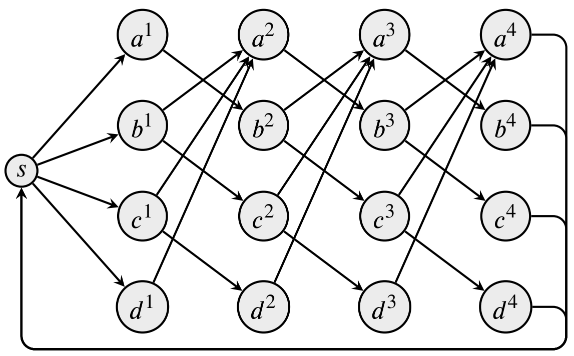

Figure 6.3: Illustration of Reduction 6.4.3, with $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$. The target sets for the disjunctive safety objective (or query) are $T_a = \{b^1, c^1, d^1, b^4, c^4, d^4\}$, $T_b = \{a^1, c^1, d^1, a^4, c^4, d^4\}$, $T_c = \{a^1, b^1, d^1, a^4, b^4, d^4\}$, and $T_d = \{a^1, b^1, c^1, a^4, b^4, c^4\}$.

(3) *The winning set of* $(G', \bigvee_{v \in V} \mathit{Safe}(T_v))$ *is non-empty.*

Proof. (1)⇒(2): Assume that G has a triangle with vertices a, b, c and let a^i, b^i, c^i be the copies of a, b, c in V^i. Now a strategy for player 1 in G' to satisfy Safe(T_a) is as follows: When at s, go to a^1; when at a^1, go to b^2; when at b^2, go to c^3; when at c^3, go to a^4; and when at a^4, go to s. As a, b, c form a triangle, all the edges required by the above strategy exist. When player 1 starts at s and follows the above strategy, the resulting play uses only the vertices s, a^1, b^2, c^3, a^4 and thus satisfies Safe(T_a).

(2)⇒(1): Assume that there is a winning play starting in s that satisfies Safe(T_a). Starting from s, at first this play has to go to a^1, as all other successors of s would violate the safety constraint. Then the play continues with some vertices $b^2 \in V^2$ and $c^3 \in V^3$ and then, again by the safety constraint, has to reach a^4. Now by the construction of G', we know that there must be edges $(a, b), (b, c), (c, a)$ in the original graph G, i.e., there is a triangle in G.

(2)⇔(3): Notice that when removing s from G', we obtain an acyclic graph and thus each infinite path has to contain s infinitely often. Thus, if the winning set is non-empty, then there is a cycle winning for some vertex and this cycle is also winning for s. For the converse direction we have that if s is in the winning set, then the winning set is non-empty. □

The graph G' defined by Reduction 6.4.3 has size linear in the size of the original graph G and can be constructed in linear time. Furthermore, we have $k = \Theta(n)$ target sets. Thus if we would have a combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm for disjunctive safety objectives or queries in graphs, we would immediately obtain a combinatorial $O(n^{3-\epsilon})$ time algorithm for triangle detection, which contradicts CTC (and thus BMM).

The above reduction uses $\Theta(n)$ safety constraints which are of size $\Theta(n)$ each. Thus, a natural question is whether smaller target sets would make the problem

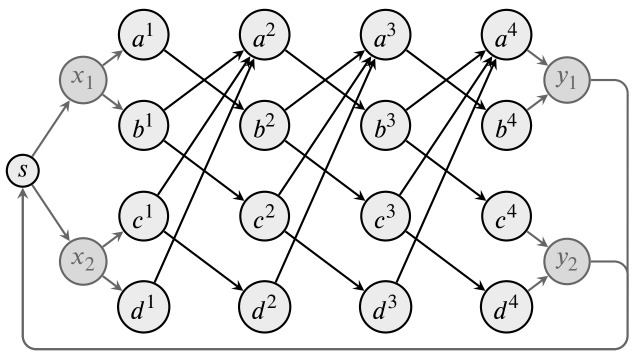

Figure 6.4: Illustration of how to reduce the number of entries in the target sets in Reduction 6.4.3 with two complete binary trees. Here $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$ and the target sets for disjunctive safety are $T_a = \{b^1, x_2, b^4, y_2\}$, $T_b = \{a^1, x_2, a^4, y_2\}$, $T_c = \{d^1, x_1, d^4, y_1\}$, and $T_d = \{c^1, x_1, c^4, y_1\}$.

any easier. We argue next that our result even holds for target sets that are of logarithmic size. To this end, we modify Reduction 6.4.3 as follows. We remove all edges incident to s and replace them by two complete binary trees. The first tree with s as root and the vertices V^1 as leaves is directed towards the leaves, the second tree with root s and leaves V^4 is directed towards s. Now for each pair v^1, v^4 one can select one vertex of each level of the trees (except for the root levels) for the set T_v such that the only safe path starting in s has to use v^1 and each safe path to s must pass v^4. As the depth of the trees is logarithmic in the number of leaf vertices, we obtain sets of logarithmic size. The construction with the binary trees is illustrated in Figure 6.4.

Next we present an $\Omega(m^{2-o(1)})$ time lower bound for disjunctive safety objectives and queries in MDPs.

Theorem 6.4.5. *There is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for disjunctive safety objectives or queries in MDPs under Conjecture 2.7.5, i.e., unless OVC and SETH fail. In particular, there is no such algorithm for deciding whether the winning set is non-empty or for deciding whether a specific vertex is winning.*

To prove the above, we give a linear time reduction from OV to disjunctive safety objectives and queries.

Reduction 6.4.6. *Given two sets S_1 and S_2 of d-dimensional vectors, we build the following MDP $P = ((V, E), (V_1, V_R))$. The target sets are given by $T_v = \{v\}$ for $v \in S_2$.*

- *The vertices V of the MDP P are given by a start vertex s, vertices S_1 and S_2 representing the sets of vectors, and vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates. The edges E of P are defined as follows: for each $x \in S_1$ there* *is an edge* to $c_i \in \mathscr{C}$ *iff* $x_i = 1$ *and for each* $y \in S_2$ *there is an edge* from $c_i \in \mathscr{C}$ *iff* $y_i = 1$*; further the start vertex* s *has an edge to every vertex of* S_1 *and every vertex* $v \in S_2$ *has an edge to* s*.*

- *The set of vertices* V *is partitioned into player 1 vertices* $V_1 = \{s\} \cup S_2$ *and random vertices* $V_R = S_1 \cup \mathscr{C}$*. Moreover, the probabilistic transition function for each vertex* $v \in V_R$ *chooses among* v*'s successors uniformly at random.*

The reduction is illustrated on an example in Figure 6.5.

Lemma 6.4.7. *Given two sets* S_1 *and* S_2 *of* d*-dimensional vectors and the corresponding MDP* P *and target sets* T_v *for* $v \in S_2$ *given by Reduction 6.4.6, the following statements are equivalent.*

(1) *There exist orthogonal vectors* $x \in S_1$ *and* $y \in S_2$*.*

(2) $s \in \bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, Safe(T_v)\right)$

(3) $s \in \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{v \in S_2} Safe(T_v)\right)$

(4) *The winning set* $\bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, Safe(T_v)\right)$ *is non-empty.*

(5) *The winning set* $\langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{v \in S_2} Safe(T_v)\right)$ *is non-empty.*

Proof. W.l.o.g., we assume that the 1-vector, i.e., the vector with all coordinates being 1, is contained in S_2 (adding the 1-vector does not change the result of the OV instance); this implies that all vertices of $\mathscr{C}$ have an outgoing edge. If the set S_1 contains the all-zero vector, an orthogonal pair of vectors exists trivially and this can be detected in linear time; thus we further assume w.l.o.g. that each vector of S_1 contains a non-zero entry and thus each vertex has an outgoing edge in P.

A play in the MDP P proceeds as follows. Starting from s, player 1 chooses a vertex $x \in S_1$; then first a vertex $c \in \mathscr{C}$ and then a vertex $y \in S_2$ are picked randomly; then the play goes back to s, starting another cycle of the play.

(1)⇒(2): Assume that there are orthogonal vectors $x \in S_1$ and $y \in S_2$. Now player 1 can satisfy $\text{Safe}(T_y)$ in the MDP P by going to x whenever the play is at s. The random player then picks some adjacent $c \in \mathscr{C}$ and then some adjacent vertex of S_2 but as x and y are orthogonal, no c reachable from x has an edge to y. Thus the play never visits y.

(2)⇒(3): Assume $s \in \bigvee_{v \in S_2} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Safe}(T_v)\right)$. Then there is a vertex $y \in S_2$ such that $s \in \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Safe}(T_y)\right)$. Now we can enlarge the objective to $\bigvee_{v \in S_2} \text{Safe}(T_v)$ and obtain $s \in \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{v \in S_2} \text{Safe}(T_v)\right)$.

(3)⇒(1): Assume $s \in \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \bigvee_{v \in S_2} \text{Safe}(T_v)\right)$ and consider a corresponding strategy σ. W.l.o.g., we can assume that this strategy is memoryless [Tho95]. Thus whenever a play is at s, the strategy picks a fixed $x \in S_1$ as the next vertex. Assume

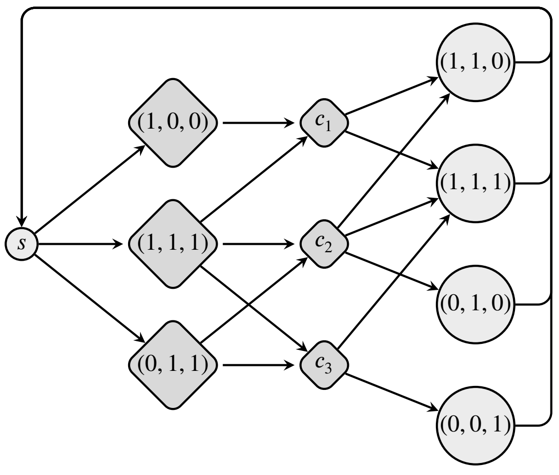

Figure 6.5: Illustration of Reduction 6.4.6, for $S_1 = \{(1,0,0),(1,1,1),(0,1,1)\}$ and $S_2 = \{(1,1,0),(1,1,1),(0,1,0),(0,0,1)\}$.

towards contradiction that there is no orthogonal vector $y \in S_2$ for x. Then for each $y \in S_2$ we have that there is a $c \in \mathscr{C}$ connecting x to y. In each cycle of a play, the play proceeds from s to x and then at random to some vertex of S_2. By the above, each of the vertices of S_2 has a non-zero probability to be reached in this cycle, which can, for each fixed n, be lower bounded by a constant p. Thus after n cycles of a play with probability at least $p^{|S_2|}$ all vertices in S_2 have been visited and thus none of the safety objectives is satisfied, a contradiction to the assumption that with probability 1 at least one safety objective is satisfied. Thus there must exist a vector $y \in S_2$ orthogonal to x.

(2)⇔(4) and (3)⇔(5): Notice that when removing s from P we get an acyclic MDP and thus each infinite path has to contain s infinitely often. Certainly if s is in the a.s. winning set, this set is non-empty. Thus let us assume there is a vertex v different from s with a winning strategy σ. All (winning) plays starting at v cross s after at most 3 steps and thus σ must also be winning when starting at s. □

The number of vertices in the MDP P defined by Reduction 6.4.6 is $O(N)$ and the number of edges m is $O(N \log N)$ (recall that $d \in \Theta(\log N)$). Furthermore, we have $k \in \Theta(N)$ target sets, and the construction can be performed in $O(N \log N)$ time. Thus, if we would have an $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm for disjunctive queries or disjunctive objectives of safety objectives for some $\epsilon > 0$, we would immediately obtain an $O(N^{2-\epsilon})$ time algorithm for OV, which contradicts OVC (and thus SETH).

6.5 MDPs with Rabin, Büchi, and co-Büchi Objectives

In the first part of this section we prove the following conditional lower bounds for Rabin, disjunctive Büchi, and disjunctive co-Büchi objectives.

Theorem 6.5.1. *Assuming CTC or BMM, there is no combinatorial $O(n^{3-\epsilon})$ or $O((kn^2)^{1-\epsilon})$ time algorithm for any $\epsilon > 0$ and any of the following problems:*

(1) *computing the a.s. winning set for an MDP with a disjunctive Büchi query;*

(2) *computing the winning set for a graph with a disjunctive co-Büchi objective and thus also computing the a.s. winning set for an MDP with a disjunctive co-Büchi objective or a disjunctive co-Büchi query;*

(3) *computing the a.s. winning set for an MDP with a Rabin objective.*

Theorem 6.5.2. *Assuming SETH or OVC, there is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm for any of the following problems:*

(1) *computing the a.s. winning set for an MDP with a disjunctive Büchi query;*

(2) *computing the a.s. winning set for an MDP with a disjunctive co-Büchi objective or a disjunctive co-Büchi query, even when each target set has cardinality one;*

(3) *computing the a.s. winning set for an MDP with a Rabin objective.*

On the algorithmic side, we prove the following theorem in the second part of this section. Note that a Rabin objective is a disjunctive objective of 1-pair Rabin objectives. The results for Büchi objectives are already implied by Observations 6.2.3 and 6.2.6, [CJH03; CH14], and Theorem 6.3.1 and are only presented for the completeness of the presentation.

Theorem 6.5.3. *We are given an MDP $P = ((V, E), (V_1, V_R), \delta)$ and a Rabin objective with target pairs $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$. Let $b = \sum_{j=1}^{k}(|L_j| + |U_j|)$ and let* MEC *denote the time to compute a MEC-decomposition.*

(1) *The almost-sure winning set $\langle\!\langle 1 \rangle\!\rangle_{as}(P, Rabin(\mathrm{TP}))$ can be computed in $O(k \cdot \textsc{mec})$ time.*

(2) *If $U_j = \emptyset$ for all $1 \leq j \leq k$, i.e., each target pair defines a Büchi objective, then the almost-sure winning set for the disjunctive objective of the 1-Pair Rabin objectives can be computed in $O(\textsc{mec} + b)$ time and for the disjunctive query in $O(k \cdot m + \textsc{mec})$ time.*

(3) *If $L_j = V$ for all $1 \leq j \leq k$, i.e., each target pair defines a co-Büchi objective, then the almost-sure winning set for the disjunctive objective and the disjunctive query of the 1-Pair Rabin objectives can computed in $O(k \cdot m + \textsc{mec})$ time.*

6.5.1 Conditional Lower Bounds for Rabin, Büchi and co-Büchi

The conditional lower bounds for Rabin and for disjunctive Büchi and co-Büchi objectives are based on our results for reachability (see Section 6.3.2) and safety objects (see Section 6.4.1) and the Observations 6.2.5, 6.2.6 & 6.2.8 that interlink these objectives. Recall that by Observation 6.2.2 disjunctive objectives and queries coincide for graphs but not MDPs.

Proposition 6.5.4. *Assuming CTC or BMM, there is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm for*

(1) *computing the winning set for an MDP with a disjunctive Büchi query,*

(2) *computing the winning set for a graph with a disjunctive co-Büchi objective, and*

(3) *computing the winning set for an MDP with a Rabin objective.*

Moreover, there is no such algorithm deciding whether the winning set is non-empty or for deciding whether a specific vertex is in the winning set.

Proof. 1) By Observation 6.2.6 in MDPs reachability queries can be reduced to Büchi queries in linear time. Thus the result follows from the corresponding conditional lower bound for reachability queries (cf. Theorem 6.3.9).

2) By Observation 6.2.5 the winning set for disjunctive safety objectives is non-empty iff the winning set for disjunctive co-Büchi objectives with the same target sets is non-empty. Thus the result follows from the corresponding conditional lower bound for safety objectives (cf. Theorem 6.4.2).
For the problem of deciding whether a specific vertex is in the winning set, recall that the graph G' constructed in Reduction 6.4.3 is such that the vertex s appears in each infinite path and thus if there is a winning strategy starting from some vertex, then there is also one starting from s. That is, deciding for G' whether s is winning is equivalent to deciding whether the winning set is non-empty.

3) The result follows from (2) and Observation 6.2.8, by which disjunctive co-Büchi objectives are special instances of Rabin objectives. □

Proposition 6.5.5. *Assuming SETH or OVC, there is no $O(m^{2-\epsilon})$ or $O((k \cdot m)^{1-\epsilon})$ time algorithm for*

(1) *computing the winning set for an MDP with a disjunctive Büchi query;*

(2) *computing the winning set for an MDP with a disjunctive co-Büchi objective or a disjunctive co-Büchi query, even when each target set has cardinality one;*

(3) *computing the winning set for an MDP with a Rabin objective.*

Moreover, there is no such algorithm for deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.

Proof. 1) By Observation 6.2.6 in MDPs reachability queries can be reduced to Büchi queries in linear time. Thus the result follows from the corresponding conditional lower bound for reachability (cf. Theorem 6.3.12).

2) By Observation 6.2.5 the winning set for a disjunctive safety objective resp. query is non-empty iff the winning set for the disjunctive co-Büchi objective resp. query with the same target sets is non-empty. Thus the result follows from the corresponding conditional lower bounds for safety objectives (cf. Theorem 6.4.5). For the problem of deciding whether a specific vertex is in the winning set, recall that the MDP P constructed in Reduction 6.4.6 is such that vertex s appears in each infinite path and thus if there is a winning strategy starting from some vertex, then there is also one starting from s. That is, deciding for P whether s is winning is equivalent to deciding whether the winning set is non-empty. Note that all target sets in Lemma 6.4.7 have cardinality one.

3) The result follows from (2) and Observation 6.2.8, by which disjunctive co-Büchi objectives are special instances of Rabin objectives. □

6.5.2 Algorithm for MDPs with Rabin Objectives

In this section we describe an algorithm for MDPs with Rabin objectives that considers each MEC of the input MDP separately. This formulation has the advantage that for the special case of disjunctive co-Büchi objectives we can then obtain a faster running time than previously known, which we describe in Section 6.5.4. The special case of disjunctive Büchi objectives is described in Section 6.5.3. The results for Büchi objectives are already implied by Observations 6.2.3 and 6.2.6, [CJH03; CH14], and Theorem 6.3.1 and are only presented for the completeness of the presentation.

For Rabin objectives a good end-component is defined as follows (see also Definition 2.6.1).

Definition 6.5.6 (Good Rabin end-component)**.** *Given an MDP P and a set* $\mathrm{TP} = \{(L_j, U_j) \mid 1 \leq j \leq k\}$ *of target pairs, a* good (Rabin) end-component *is an end-component X of P such that $L_j \cap X \neq \emptyset$ and $U_j \cap X = \emptyset$ for some $1 \leq j \leq k$.*

As for Streett objectives, we determine the almost-sure winning set for Rabin objectives by computing the almost-sure winning set for the reachability objective with the union of all good Rabin end-components as target set. The correctness of this approach follows from Corollary 2.6.5 and Proposition 2.6.9. Our strategy to find all good Rabin end-components is as follows. First the MEC-decomposition of the input MDP P is determined. For each MEC X, and separately for each $1 \leq j \leq k$, we first remove the set U_j and its random attractor and then compute the MEC-decomposition in the sub-MDP induced by the remaining vertices. Every newly computed MEC that contains a vertex of L_j is a good Rabin end-component. If the MEC X of P contains one such good end-component, then by Corollary 2.6.6 all vertices of X are in the almost-sure winning set for the Rabin objective. Thus we can immediately add X to the set of winning MECs in line 8.[2]

[2] We could alternatively add only the vertices in the good end-component because the winning

Algorithm RabinMDP: Algorithm for MDPs with Rabin objectives

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target pairs $\mathrm{TP} = \{(L_j, U_j) \mid 1 \le j \le k\}$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \mathrm{Rabin(TP)})$
1  $\mathcal{X} \leftarrow$ allMECs($P$); winMEC $\leftarrow \emptyset$
2  foreach $X \in \mathcal{X}$ do
3      for $1 \le j \le k$ do
4          if $L_j \cap X \ne \emptyset$ then
5              $\mathcal{Y} \leftarrow$ allMECs($X \setminus Attr_R(P[X], U_j)$)
6              foreach $Y \in \mathcal{Y}$ do
7                  if $L_j \cap Y \ne \emptyset$ then
8                      winMEC $\leftarrow$ winMEC $\cup \{X\}$
9                      continue with next $X \in \mathcal{X}$
10 return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \mathrm{winMEC}} X)\right)$
```

Proposition 6.5.7 (Running time)**.** *Algorithm RabinMDP can be implemented in $O(k \cdot \textsc{mec})$ time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P can clearly be done in $O(\textsc{mec})$ time. Further, by Theorem 6.3.1, the final almost-sure reachability computation can be done in $O(\textsc{mec})$ time[3]. Assume that each vertex has a list of the sets L_j and U_j for $1 \le j \le k$ it belongs to; we can generate these lists from the lists of the target pairs in $O(b) \in O(nk)$ time at the beginning of the algorithm. Consider an iteration of the outer for-each loop, let X denote the considered MEC, and fix one iteration j of the k iterations of the for-loop. Determining the intersection $L_j \cap X$ in line 4 can be done in $O(|X|)$ time. Let m_X be the number of edges in $P[X]$ and let $\textsc{mec}_X$ denote the time needed to compute a MEC-decomposition on $P[X]$. Line 5 requires $O(m_X + \textsc{mec}_X) = O(\textsc{mec}_X)$ time. The inner for-each loop takes $O(|X|)$ time as in each iteration $O(|Y|)$ time for the intersection in line 7 and constant time for line 8 are sufficient. Thus in total we have $O(b + \textsc{mec} + \sum_{X \in \mathcal{X}} k \cdot (|X| + \textsc{mec}_X)) \in O(k \cdot \textsc{mec})$. □

Proposition 6.5.8 (Correctness)**.** *Algorithm RabinMDP computes the almost-sure winning set $\langle\!\langle 1 \rangle\!\rangle_{as}(P, Rabin(\mathrm{TP}))$.*

Proof. By the Corollaries 2.6.6 and 2.6.5 and Proposition 2.6.9 we know that it suffices to correctly classify each MEC as either *winning* or *not winning*; we say a MEC is *winning* iff it contains a good Rabin end-component, that is, it contains an

MEC would be detected as winning in the final almost-sure reachability computation; the presented formulation shows the similarities to the co-Büchi algorithm in Section 6.5.4. Additionally, this allows reusing the initial MEC-decomposition for the almost-sure reachability computation.

[3]Actually the almost-sure reachability computation can be done in $O(m)$ by reusing the already computed MEC-decomposition.

end-component X such that $L_j \cap X \neq \emptyset$ and $U_j \cap X = \emptyset$ for some $1 \leq j \leq k$. The loops in lines 2 and 3 iterate over all MECs X and all target pairs (L_j, U_j). It remains to show that lines 4–8 correctly classify whether a MEC contains a good end-component X' with $L_j \cap X' \neq \emptyset$ and $U_j \cap X' = \emptyset$.

- Assume X contains a good end-component X' with $L_j \cap X' \neq \emptyset$ and $U_j \cap X' = \emptyset$. Then the if condition in line 4 is true and, in the j-th iteration of the for-loop, the algorithm removes the random attractor of U_j from X and computes the MECs $\mathcal{Y}$ of the MDP induced by the remaining vertices of X. As X' is strongly connected, has no outgoing random edges, and $U_j \cap X' = \emptyset$, it does not intersect with $Attr_R(P[X], U_j)$ by Lemma 2.6.10. Thus there is a MEC $Y \in \mathcal{Y}$ that contains X' and thus has $L_j \cap Y \neq \emptyset$. Hence, the algorithm correctly classifies the set X as a winning MEC.

- Assume the algorithm classifies a MEC X as winning. Then for some j in line 7 there is an end-component $Y \in \mathcal{Y}$ of $P[X \setminus Attr_R(P[X], U_j)]$ with $L_j \cap Y \neq \emptyset$ and $U_j \cap Y = \emptyset$, i.e., Y is a good end-component in $P[X \setminus Attr_R(P[X], U_j)]$. Moreover, there cannot be a random edge from $u \in Y$ to $Attr_R(P[X], U_j)$ as such an u would be included in the random attractor $Attr_R(P[X], U_j)$; and by definition of a MEC, X has no outgoing random edges. Thus Y is also a good end-component of the full MDP P, i.e., X is classified correctly.

By the above we have that whenever the outer for-each loop terminates, the set winMEC consists of all winning MECs and then by Corollary 2.6.5 and Proposition 2.6.9 we can compute $\langle\!\langle 1 \rangle\!\rangle_{as}(P, \text{Rabin}(\text{TP}))$ by computing the almost-sure winning set for the reachability objective with the union of all winning MECs as target set. □

6.5.3 Algorithms for MDPs with Büchi Objectives

As Büchi objectives can be encoded by one-pair Rabin objectives, Algorithm RabinMDP can also be used to compute the a.s. winning set for disjunctive Büchi objectives. However, Büchi objectives allow for some immediate simplifications that result in Algorithm DisjObjBuchiMDP. This simplifications are based on the observation that for disjunctive Büchi objectives all sets U_j are empty and therefore also the random attractors computed in line 5 of Algorithm RabinMDP are empty. Hence, there is also no need to recompute the MECs, and therefore deciding whether a MEC is winning reduces to testing whether it intersects with one of the target sets.

Proposition 6.5.9 (Running time). *Algorithm DisjObjBuchiMDP can be implemented in $O(\textsc{mec} + b)$ time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P and, by Theorem 6.3.1, the final almost-sure reachability computation can be done in $O(\textsc{mec})$

Algorithm DisjObjBuchiMDP: MDPs with disjunctive Büchi objectives

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target sets $T_j$ for $1 \leq j \leq k$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \bigvee_{1 \leq j \leq k} \text{Büchi}(T_j)\right)$
1 $\mathcal{X} \leftarrow \text{ALLMECs}(P)$; winMEC $\leftarrow \emptyset$; $T \leftarrow \bigcup_{1 \leq j \leq k} T_j$
2 foreach $X \in \mathcal{X}$ do
3     if $T \cap X \neq \emptyset$ then
4         winMEC $\leftarrow$ winMEC $\cup \{X\}$
5 return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \text{winMEC}} X)\right)$
```

time. Assume that each vertex has a flag indicating whether it is in one of the sets T_j or in none of them; we can generate these flags from lists of the sets T_j in $O(b)$ time at the beginning of the algorithm. Consider an iteration of the for-each loop, let X denote the considered MEC, and fix some iteration j of the for-loop. One iteration costs $O(|X|)$ time because in each iteration the intersection $T \cap X$ in line 3 can be determined in $O(|X|)$ time and line 4 takes constant time. Thus in total the algorithm runs in $O(\text{MEC} + n + b) \in O(\text{MEC} + b)$ time. □

Next we describe the algorithm for disjunctive Büchi queries with k target sets T_j. Recall that by definition a vertex is in the a.s. winning set for a disjunctive query if it is in the a.s. winning set for at least one of the objectives, i.e., we can determine the a.s. winning set for the disjunctive query by taking the union of the a.s. winning sets for each of the single objectives. We say that a MEC X is *winning for a target set T_j* w.r.t. the Büchi objective defined by T_j if $T_j \cap X \neq \emptyset$. Algorithm DisjQueryBuchiMDP first determines the winning MECs, denoted by winMEC_j, for each of the target sets T_j separately. However, as the MEC-decomposition is independent of the sets T_j, it suffices to compute the MEC-decomposition once. The a.s. winning set for the disjunctive Büchi query is then computed via a disjunctive reachability query, where the j-th target set is given by the union of the winning MECs for T_j. Using Algorithm DisjReachMDP, the disjunctive reachability query takes time $O(k \cdot m + \text{MEC})$ (Theorem 6.3.1). Thus the only differences to Algorithm DisjObjBuchiMDP are that (a) the winning MECs are determined and stored separately for each T_j with $1 \leq j \leq k$ and (b) a disjunctive reachability query with k target sets is performed in the final step of the algorithm, while in Algorithm DisjObjBuchiMDP it is sufficient to consider a single target set for almost-sure reachability. This immediately results in a running time of $O(k \cdot m + \text{MEC} + b) = O(k \cdot m + \text{MEC})$.

Corollary 6.5.10 (Running time)**.** *Algorithm DisjQueryBuchiMDP can be implemented in $O(k \cdot m + \text{MEC})$ time.*

Algorithm DisjQueryBuchiMDP: MDPs with disjunctive Büchi queries

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target sets $T_j$ for $1 \leq j \leq k$
Output: $\bigvee_{1 \leq j \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{Büchi}(T_j)\right)$
1 $\mathcal{X} \leftarrow$ allMECs($P$)
2 for $1 \leq j \leq k$ do
3     $\mathsf{winMEC}_j \leftarrow \emptyset$
4 foreach $X \in \mathcal{X}$ do
5     for $1 \leq j \leq k$ do
6         if $T_j \cap X \neq \emptyset$ then
7             $\mathsf{winMEC}_j \leftarrow \mathsf{winMEC}_j \cup \{X\}$
8 return $\bigvee_{1 \leq j \leq k} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, Reach(\bigcup_{X \in \mathsf{winMEC}_j} X)\right)$
```

6.5.4 Algorithms for MDPs with co-Büchi Objectives

As co-Büchi objectives can be encoded by one-pair Rabin objectives, Algorithm RabinMDP can be used to compute the a.s. winning set for disjunctive co-Büchi objectives. However, co-Büchi objectives allow for some simplifications that result in the simpler and more efficient Algorithm DisjObjCoBuchiMDP. These simplifications are based on the observation that for disjunctions of co-Büchi objectives all sets L_j coincide with the set of all vertices and therefore the if-conditions in lines 4 and 7 of Algorithm RabinMDP are always true. That is, whenever there is a vertex in a MEC X of P that is not contained in $Attr_R(P[X], T_j)$, then there is a MEC in $P[X \setminus Attr_R(P[X], T_j)]$, which is a good end-component of P. Testing whether a MEC contains a good end-component for a co-Büchi objective coBüchi(T_j) thus reduces to testing whether the random attractor of T_j covers the whole MEC.

Observation 6.5.11. *For the disjunction of one-pair Streett objectives the same ideas can be used (Table 6.5). For each MEC X and each j we check whether $X \cap L_j \neq \emptyset$ and $X \cap U_j = \emptyset$. If this is the case, then we determine whether the random attractor of L_j covers the whole MEC. If the latter does not hold, then the MEC contains a good end-component for the one-pair Streett objective.*

Proposition 6.5.12 (Running time)**.** *Algorithm DisjObjCoBuchiMDP can be implemented in $O(k \cdot m + \textsc{mec})$ time.*

Proof. The initialization of $\mathcal{X}$ with all MECs of the input MDP P and, by Theorem 6.3.1, the final almost-sure reachability computation can be done in $O(\textsc{mec})$ time. Consider an iteration of the for-each loop, let X denote the considered MEC, and fix some iteration j of the for-loop. Let m_X be the number of edges in $P[X]$. In the j-th iteration we need $O(|m_X|)$ time to compute the random attractor in line 4 and constant time in line 5. Thus the total time is $O(k \cdot m + \textsc{mec})$. □

Algorithm DisjObjCoBuchiMDP: MDPs with disjunctive co-Büchi obj.

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target sets $T_j$ for $1 \le j \le k$
Output: $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, \bigvee_{1 \le j \le k} \text{coBüchi}(T_j)\right)$
1 $\mathcal{X} \leftarrow$ allMECs($P$); winMEC $\leftarrow \emptyset$
2 foreach $X \in \mathcal{X}$ do
3     for $1 \le j \le k$ do
4         if $X \not\subseteq Attr_R(P[X], T_j)$ then
5             winMEC $\leftarrow$ winMEC $\cup \{X\}$
6             continue with next $X \in \mathcal{X}$
7 return $\langle\!\langle 1 \rangle\!\rangle_{as}\left(P, Reach(\bigcup_{X \in \text{winMEC}} X)\right)$
```

Recall that we can determine the a.s. winning set for a disjunctive *query* by taking the union of the a.s. winning sets for each of the single objectives. Thus for disjunctive co-Büchi queries with target sets T_j for $1 \le j \le k$, we consider the target sets T_j separately. However, as the MEC-decomposition is independent of the target sets, it suffices to compute the MEC-decomposition once. We say that a MEC X is *winning for a target set* T_j w.r.t. the co-Büchi objective specified by T_j if $X \not\subseteq Attr_R(P[X], T_j)$. We test whether this is satisfied in l. 6 of Algorithm DisjQueryCoBuchiMDP. Whenever a MEC is winning for some T_j, it is added to winMEC$_j$. The a.s. winning set for the disjunctive co-Büchi query can then be determined by computing the a.s. winning set for a disjunctive query of k reachability objectives, where the target set for the j-th reachability objective is the union of the MECs in winMEC$_j$. Compared to Algorithm DisjObjCoBuchiMDP, this increases the running time for the almost-sure reachability computation to $O(k \cdot m)$ (given the MEC-decomposition), which, however, is subsumed by the total running time of $O(k \cdot m + \text{MEC})$. The resulting algorithm is stated as Algorithm DisjQueryCoBuchiMDP.

Corollary 6.5.13 (Running time)**.** *Algorithm DisjQueryCoBuchiMDP can be implemented in $O(k \cdot m + \text{MEC})$ time.*

6.6 Algorithm for Graphs with Singleton co-Büchi Objectives

In this section we show how to compute in linear time the winning set for graphs with a special type of disjunctive co-Büchi objective (or, equivalently, query), namely when all sets T_j for $1 \le j \le k$ have cardinality one. We assume w.l.o.g. that the sets T_j are pairwise disjoint.

Algorithm DisjQueryCoBuchiMDP: MDPs with disj. co-Büchi queries

```
Input  : MDP $P = ((V, E), (V_1, V_R), \delta)$ and
         target sets $T_j$ for $1 \le j \le k$
Output: $\bigvee_{1 \le j \le k} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, \text{coBüchi}(T_j)\right)$
1 $\mathcal{X} \leftarrow$ allMECs($P$)
2 for $1 \le j \le k$ do
3     $\mathsf{winMEC}_j \leftarrow \emptyset$
4 foreach $X \in \mathcal{X}$ do
5     for $1 \le j \le k$ do
6         if $X \not\subseteq Attr_R(P[X], T_j)$ then
7             $\mathsf{winMEC}_j \leftarrow \mathsf{winMEC}_j \cup \{X\}$
8 return $\bigvee_{1 \le j \le k} \langle\!\langle 1 \rangle\!\rangle_{as} \left(P, Reach(\bigcup_{X \in \mathsf{winMEC}_j} X)\right)$
```

Theorem 6.6.1. *Given a graph $G = (V, E)$ and co-Büchi objectives T_j with $|T_j| = 1$ for $1 \le j \le k$, the winning set for the disjunctive co-Büchi objective (or, equivalently, query) can be computed in $O(m)$ time.*

We call an SCC *winning* if it contains a good component (see Section 2.6.1), i.e., a strongly connected subgraph induced by a set of vertices S such that a play ω with $\text{Inf}(\omega) = S$ satisfies the objective. To compute the winning set for disjunctive singleton co-Büchi objectives, it is sufficient to detect whether a strongly connected graph contains a cycle that does *not contain all* the vertices in the set $T = \bigcup_{1 \le j \le k} T_j$. To see this, first note that each non-trivial SCC of the graph (i.e., each SCC that contains at least one edge) that does not contain all vertices of T is winning. If there is no SCC S with $T \subseteq S$, then we can determine the winning set in linear time by computing the vertices that can reach any non-trivial SCC. Thus it remains to consider an SCC S with $T \subseteq S$. For the relevant case of $|T| > 1$ we have that S is a non-trivial SCC. Since S is strongly connected, the vertices of S can reach each other and hence it is sufficient to compute whether S contains a cycle that does not contain all the vertices of T (i.e., to solve the *non-emptiness* problem). If such a cycle exists, then S is winning, otherwise S is not winning. In any case, the winning set can then be determined by computing the vertices that can reach some winning SCC.

We now describe the algorithm to determine whether a strongly connected graph $G = (V, E)$ contains a simple cycle C such that we have $T_j \cap C = \emptyset$ for some $1 \le j \le k$, given $|T_j| = 1$ for all j. First we check whether $G[V \setminus T_1]$ contains a non-trivial SCC (l. 3). If this is true, then G contains a cycle that does not contain T_1 and we are done. Otherwise every cycle of G contains T_1. We assign the edges of G edge lengths as follows: All edges $(v, w) \in E$ for which $w \in T$ have length 1, all other edges have length 0. Let s denote the vertex in T_1. Let δ be the length of the shortest path (w.r.t. the edge lengths defined above) from s to s that uses at

Algorithm SingletonCBGraph: Disjunctive singleton co-Büchi on graphs

```
Input  : strongly connected graph $G = (V, E)$ and
         disjoint target sets $T_j$ with $|T_j| = 1$ for $1 \leq j \leq k$ and $k > 1$
Output: "yes" if there is a cycle $C$ with $T \not\subseteq C$; "no" otherwise
1  let $T = \bigcup_j T_j$
2  $\mathscr{C} \leftarrow$ allSCCs($G[V \setminus T_1]$)
3  if $\mathscr{C}$ contains non-trivial SCC then
4      return yes
5  else
6      let $s$ be the vertex in $T_1$
7      replace $s$ with $s_{\text{in}}$ and $s_{\text{out}}$: $s_{\text{in}}$ gets in-edges and $s_{\text{out}}$ gets out-edges of $s$
8      $Q_0 \leftarrow \{s_{\text{out}}\}$; mark $s_{\text{out}}$
9      for $t \leftarrow 0$ to $k-1$ do
10         $Q_{t+1} \leftarrow \emptyset$
11         while $Q_t \neq \emptyset$ do
12             remove $v$ from $Q_t$
13             if $v = s_{in}$ then
14                 return yes
15             foreach $(v, w) \in E$ with $w$ not marked do
16                 mark $w$
17                 if $w \in T$ then add $w$ to $Q_{t+1}$
18                 else add $w$ to $Q_t$
19     return no
```

least one edge, i.e., the minimum length of a cycle containing s. We have that $\delta < k$ if and only if this cycle with the length δ does not contain all vertices of T. Thus if $\delta < k$, then G is winning for the disjunctive co-Büchi objective, otherwise not. Note that this algorithm would also work for a Rabin objective where we have for each $1 \leq j \leq k$ that (a) $L_j = \{s\}$ for some $s \in V$ and (b) $|U_j| = 1$.

Since all edge lengths are zero or one, we can compute δ in linear time. In Algorithm SingletonCBGraph we additionally use that all incoming edges of a vertex have the same length. After checking whether $G[V \setminus T_1]$ contains a non-trivial SCC, the algorithm works as follows. We modify the graph by replacing the vertex s by two vertices, s_{in} and s_{out}, and replacing s in all edges $(v, s) \in E$ with s_{in} and in all edges $(s, v) \in E$ with s_{out} (l. 7). Then δ is equal to the shortest path from s_{out} to s_{in}. For the algorithm we consider both s_{in} and s_{out} to be contained in T. In the t-th iteration of the for-loop we consider two "queues", Q_t and Q_{t+1} (can be implemented as sets). Each vertex is added to a queue at most once during the algorithm, which is ensured by marking vertices when they are added to a queue (l. 16) and only add before unmarked vertices. The following lemma shows that, until the vertex s_{in} is removed from Q_t and the algorithm terminates, precisely the vertices with distance t from s_{out} are added to Q_t for each t. Thus s_{in} is added to Q_t for some $t < k$

if and only if $\delta < k$, which shows the correctness of the algorithm. The running time of the algorithm is $O(m)$ because each vertex is added to and removed from a queue at most once and thus the outgoing edges of a vertex are only examined once, namely when it is removed from a queue (l. 12).

Lemma 6.6.2. *Before each iteration t of the for-loop in Algorithm SingletonCBGraph, with t ranging from 0 to $k-1$, the queue Q_t contains the vertices of T with distance t from s_{out}. During iteration t, the vertices of $V \setminus T$ with distance t from s_{out} are added to Q_t. No other vertices are added to Q_t.*

Proof. The proof is by induction over the iterations of the for-loop. Before the first iteration ($t = 0$), Q_0 is initialized with s_{out} and all queues Q_t for $t > 0$ are empty, thus the induction base holds. Assume the claim holds before the t-th iteration. At the end of the while-loop, Q_t is empty; every vertex v that has been added to Q_t before or in the t-th iteration of the for-loop is removed from Q_t in some iteration of the while-loop. Then all the unmarked vertices w with $(v, w) \in E$ are marked and added to Q_t if the edge (v, w) has length zero or added to Q_{t+1} if the edge (v, w) has length one. A vertex $u \in V \setminus T$ with distance at least t from s_{out} has distance exactly t if and only if it can be reached from some vertex $v \in T$ that has distance t by a sequence of zero length edges. The while-loop precisely adds these vertices to Q_t. Further, a vertex $u \in V \cap T$ has distance $t + 1$ if and only if it has an edge from some vertex $v \in V$ that has distance t. The while-loop adds exactly these vertices to Q_{t+1}. □

6.7 Conclusion

In this chapter we present new algorithms and the first conditional super-linear lower bounds for several fundamental model-checking problems in graphs and MDPs w.r.t. to ω-regular objectives. Our results establish the first model separation results for graphs and MDPs w.r.t. to classical ω-regular objectives, and the first objective separation results both in graphs and MDPs for dual objectives, and the conjunction and disjunction of objectives of the same type. An interesting direction for future work is to consider similar results for other models, such as, graphs games. First results in this direction can be found in Chapter 7.

For sparse MDPs with Rabin objectives it remains open to close the gap between the $O(km^{1.5})$ upper bound and the $\Omega((km)^{1-o(1)})$ conditional lower bound. Furthermore, it would be interesting to remove the “combinatorial” assumption from the conditional lower bounds for dense graphs, or show that fast matrix multiplication can be used to obtain faster algorithms.

CHAPTER 7

Generalized Büchi and Generalized Reactivity-1 Games

7.1 Introduction

In this chapter we consider game graphs with generalized Büchi and GR(1) objectives. Since for reactive systems there are multiple requirements, the conjunction of Büchi objectives, which is known as *generalized Büchi objective*, is a very central objective to study for game graphs. Generalized Büchi objectives are required to specify progress conditions of mutual exclusion protocols, and deterministic Büchi automata can express many important properties of linear-time temporal logic (LTL) (the de-facto logic to specify properties of reactive systems) [KV05; KV98; AT04; KPB94]. The analysis of reactive systems with such objectives naturally gives rise to generalized Büchi games.

GR(*1*) (*generalized reactivity* (*1*)) objectives are currently a very popular class of objectives to specify behaviors of reactive systems [PPS06]. A GR(1) objective is an implication between two generalized Büchi objectives: the antecedent generalized Büchi objective is called the assumption and the consequent generalized Büchi objective is called the guarantee. In other words, the objective requires that if the assumption generalized Büchi objective is satisfied, then the guarantee generalized Büchi objective must also be satisfied. Game graphs with GR(1) objectives have been used in many applications, such as the industrial example of synthesis of AMBA AHB protocol [Blo+07; GCH13] as well as in robotics applications [FKP05; Cha+15a].

For the problems we consider, while polynomial-time algorithms are known, there are no super-linear lower bounds. Since for polynomial-time algorithms unconditional super-linear lower bounds are very rare in the whole of computer science, we consider *conditional lower bounds*, which assume that for some well-studied problem the known algorithms are optimal up to some lower-order factors. We consider two such well-studied assumptions, (A1) the combinatorial Boolean matrix multiplication conjecture (BMM) and (A2) the Strong Exponential Time Hypothesis (SETH). The conjectures are formally stated in Section 2.7, see Chapter 1 for their relevance and other conjectures.

We will distinguish between results most relevant for *sparse graphs*, where the number of edges m is roughly proportional to the number of vertices n, and *dense graphs* with $m = \Theta(n^2)$. Sparse graphs arise naturally in program verification, as control-flow graphs are sparse [Tho98; Cha+15b]. Graphs obtained as synchronous product of several components (where each component makes transitions at each step) [KP09; Cha+16d] can lead to dense graphs.

Recall from Chapters 1 and 2 that the basic algorithmic problem is to compute the winning set, i.e., the set of starting vertices where player 1 can ensure the objective irrespective of the way player 2 plays; the way player 1 achieves this is called her winning strategy. We specify a game by a game graph $\mathcal{G}$ and an objective ϕ for player 1. Player 2 has the complementary objective $\Omega \setminus \phi$.

Our results. We consider game graphs with n vertices and m edges with generalized Büchi objectives with k conjunctions, and target sets of size $b_1, b_2, \dots, b_k$, and GR(1) objectives with k_1 conjunctions in the assumptions and k_2 conjunctions in the guarantee. Below we state our results in relation to existing work.

- *Generalized Büchi objectives.* The classical algorithm for generalized Büchi objectives requires $O(k \cdot \min_{1 \le \ell \le k} b_\ell \cdot m)$ time. Furthermore, there exists an $O(k^2 \cdot n^2)$ time algorithm via a reduction to Büchi games [Blo+10; CH14].

  (1) *Dense graphs.* Since $\min_{1 \le \ell \le k} b_\ell = O(n)$ and $m = O(n^2)$, in terms of n and k the classical algorithm has a worst-case running time of $O(k \cdot n^3)$. First, we present an algorithm with worst-case running time $O(k \cdot n^2)$, which is an improvement for instances with $\min_{1 \le \ell \le k} b_\ell \cdot m = \omega(n^2)$. Second, for dense graphs with $m = \Theta(n^2)$ and $k = \Theta(n^c)$ for any $0 < c \le 1$ our algorithm is optimal under (A1); i.e., improving our algorithm for dense graphs would imply a sub-cubic combinatorial Boolean matrix multiplication algorithm.

  (2) *Sparse graphs.* We show that for $k = \Theta(n^c)$ for any $0 < c \le 1$, for target sets of constant size, and sparse graphs with $m = \Theta(n^{1+o(1)})$ the basic algorithm is optimal under (A2). In fact, our conditional lower bound under (A2) holds even when each target set is a singleton. Quite strikingly, our result implies that improving the basic algorithm for sparse graphs even with singleton sets would imply a major breakthrough in overcoming the exponential barrier for SAT.

  In summary, for games on graphs, we present an improved algorithm for generalized Büchi objectives for dense graphs that is optimal under (A1); and show that under (A2) the basic algorithm is optimal for sparse graphs and constant size target sets.

The conditional lower bound for dense graphs means in particular that for unrestricted inputs the dependence of the running time on n cannot be improved, whereas the bound for sparse graphs makes the same statement for the dependence on m. Moreover, as the graphs in the reductions for our lower bounds can be made acyclic by deleting a single vertex, our lower bounds also apply to a broad range of digraph parameters. For instance, let w be the DAG-width [Ber+06] of a graph, then there is no $O(f(w) \cdot (k \cdot n^2)^{1-o(1)})$ time algorithm (A1) and no $O(f(w) \cdot (km)^{1-o(1)})$ time algorithm under (A2).

- *GR(1) objectives.* We present an algorithm for games on graphs with GR(1) objectives that has $O(k_1 \cdot k_2 \cdot n^{2.5})$ running time and improves the previously known $O(k_1 \cdot k_2 \cdot m \cdot n)$ time algorithm [JP06] for $m \in \omega(n^{1.5})$. Note that since generalized Büchi objectives are special cases of GR(1) objectives, our conditional lower bounds for generalized Büchi objectives apply to GR(1) objectives as well, but are not tight.

All our algorithms can easily be modified to also return the corresponding winning strategies for both players within the same time bounds.

Comparison with related models. We compare our results for game graphs to the special case of (standard) graphs (i.e., games with only player 1) and the related model of Markov decision processes (MDPs) (with player 1 and stochastic transitions). First note that for reachability objectives, linear-time algorithms exist for game graphs [Bee80; Imm81], whereas for MDPs (for the computation of the almost-sure winning set) the best-known algorithm has a running time of $O(\min(n^2, m^{1.5}))$ [CJH03; CH14]. For MDPs with reachability objectives, a linear or even $O(m \log n)$ time algorithm is a major open problem, i.e., there exist problems that seem harder on MDPs than on game graphs. Our conditional lower bound results show that under assumptions (A1) and (A2) the algorithmic problem for generalized Büchi objectives is strictly harder on game graphs as compared to standard graphs and MDPs. More concretely, for $k = \Theta(n)$, (a) for dense graphs ($m = \Theta(n^2)$) and $\min_{1 \leq \ell \leq k} b_\ell = \Omega(\log n)$, our lower bound for games on graphs under (A2) is $\Omega(n^{3-o(1)})$, whereas both the graph and the MDP problems can be solved in $O(n^2)$ time [CH14]; and (b) for sparse graphs ($m = \Theta(n^{1+o(1)})$) with $\min_{1 \leq \ell \leq k} b_\ell = O(1)$, our lower bound for games on graphs under (A1) is $\Omega(m^{2-o(1)})$, whereas the graph problem can be solved in $O(m)$ time and the MDP problem in $O(m^{1.5})$ time [AH04; CH14]; respectively.

Outline. In Section 7.2 we consider algorithms for generalized Büchi objectives and first present a basic algorithm which is in $O(kmn)$ time and then improve it to an $O(k \cdot n^2)$ time algorithm. In Section 7.3 we provide conditional lower bounds for generalized Büchi objectives. Finally, in Section 7.4 we study algorithms for games with GR(1) objective and first give a basic algorithm which is in $O(k_1 \cdot k_2 \cdot n^3)$ time (using the $O(k \cdot n^2)$ time algorithm for generalized Büchi games as a subroutine) and then improve it to an $O(k_1 \cdot k_2 \cdot n^{2.5})$ time algorithm.

7.2 Algorithms for Generalized Büchi Games

For generalized Büchi games we first present the basic algorithm that follows from the results of [EJ91; McN93; Zie98]. The basic algorithm (cf. Algorithm GenBuchiBasic) runs in time $O(kmn)$. We then improve it to an $O(k \cdot n^2)$ time algorithm by exploiting ideas from the $O(n^2)$ time algorithm for Büchi games in [CH14]. The basic algorithm is fast for instances where one target set is small, i.e., the algorithm runs in time $O(k \cdot min_{1 \le \ell \le k} b_\ell \cdot m)$ time, where $b_\ell = |T_\ell|$.

Reduction to Büchi games. Another way to implement generalized Büchi games is by a reduction to Büchi games as follows (see also [Blo+10]). Make k copies V^ℓ, $1 \le \ell \le k$, of the vertices of the original game graph and draw an edge (v^ℓ, u^ℓ) if (v, u) is an edge in the original graph and $v \notin T_\ell$, and an edge $(v^\ell, u^{\ell \oplus 1})$ if (v, u) is an edge in the original graph and $v \in T_\ell$ (where $\ell \oplus 1 = \ell + 1$ for $\ell < k$ and $k \oplus 1 = 1$). Finally, pick the target set T_ℓ of minimal size and make its copy T_ℓ^ℓ in V^ℓ the target set for the Büchi game. This reduction results in another $O(k \cdot min_{1 \le \ell \le k} b_\ell \cdot m)$ time algorithm when combined with the basic algorithm for Büchi games and in an $O(k^2 n^2)$ time algorithm when combined with the $O(n^2)$ time algorithm for Büchi games [CH14].

Notation. Our algorithms iteratively identify sets of vertices that are winning for player 2, i.e., player-2 dominions, and remove them from the graph. In the algorithms and their analysis we denote the sets in the t-th-iteration with superscript t, in particular $\mathcal{G}^1 = \mathcal{G}$, where $\mathcal{G}$ is the input game graph, G^t is the graph of $\mathcal{G}^t$, V^t is the vertex set of G^t, and $T_\ell^t = V^t \cap T_\ell$. We also use $\{T_\ell^t\}$ to denote the list of target sets $(T_1^t, T_2^t, \ldots, T_k^t)$, in particular when updating all the sets in a uniform way.

7.2.1 Basic Algorithm

In this subsection we establish the foundations for our improved algorithm for generalized Büchi games by providing a proof of the following folklore theorem.

Theorem 7.2.1 (folklore). *The winning sets for generalized Büchi games with n vertices and k target sets can be computed in time* $O(k \cdot \min_{1 \le \ell \le k} b_\ell \cdot m) \in O(kmn)$.

The intuition behind the basic algorithm for generalized Büchi games is as follows. Recall that for a player-1 closed set U, player 2 has a strategy from each vertex $u \in U$ that ensures that the play never leaves U (see Section 2.5.3). Thus, if for a 1-closed set U there is a target set T_ℓ with $T_\ell \cap U = \emptyset$, then the set U is a player-2 dominion. Moreover, if U is a player-2 dominion, also the attractor $Attr_2(\mathcal{G}, U)$ of

U is a player-2 dominion. The basic algorithm (cf. Algorithm GenBuchiBasic) proceeds as follows. It iteratively computes vertex sets S^t closed for player 1 that do not intersect with one of the target sets. If such a player-2 dominion S^t is found, then all vertices of $Attr_2(\mathcal{G}^t, S^t)$ are marked as winning for player 2 and removed from the game graph. The remaining game graph is denoted by $\mathcal{G}^{t+1}$. To find a player-2 dominion S^t, for each $1 \le \ell \le k$ the attractor $Y^t_\ell = Attr_1(\mathcal{G}^t, T^t_\ell)$ of the target set T^t_ℓ is determined. If for some ℓ the complement of Y^t_ℓ is not empty, then we assign $S^t = V^t \setminus Y^t_\ell$ for the smallest such ℓ. The algorithm terminates if in some iteration t for each $1 \le \ell \le k$ the attractor Y^t_ℓ contains all vertices of V^t. In this case the set V^t is returned as the winning set of player 1. The winning strategy of player 1 from these vertices is then a combination of the attractor strategies to the sets T^t_ℓ.

Algorithm GenBuchiBasic: Generalized Büchi games in $O(k \cdot b_1 \cdot m)$ time

```
Input  : game graph $\mathcal{G} = (G, (V_1, V_2))$ with graph $G = (V, E)$ and
         generalized Büchi objective $\bigwedge_{1 \le \ell \le k}$ Büchi$(T_\ell)$
Output : winning set of player 1
1  $\mathcal{G}^1 \leftarrow \mathcal{G}$
2  $\{T^1_\ell\} \leftarrow \{T_\ell\}$
3  $t \leftarrow 0$
4  repeat
5      $t \leftarrow t + 1$
6      for $1 \le \ell \le k$ do
7          $Y^t_\ell \leftarrow Attr_1(\mathcal{G}^t, T^t_\ell)$
8          $S^t \leftarrow V^t \setminus Y^t_\ell$
9          if $S^t \neq \emptyset$ then break
10     $D^t \leftarrow Attr_2(\mathcal{G}^t, S^t)$
11     $\mathcal{G}^{t+1} \leftarrow \mathcal{G}^t \setminus D^t$
12     $\{T^{t+1}_\ell\} \leftarrow \{T^t_\ell \setminus D^t\}$
13 until $D^t = \emptyset$
14 return $V^t$
```

Proposition 7.2.2 (Running time). *Algorithm GenBuchiBasic terminates in $O(k \cdot b_1 \cdot m)$ time, where $b_1 = |T_1|$, and thus also in $O(kmn)$ time .*

Proof. In each iteration of the repeat-until loop at most $k+1$ attractor computations are performed, each of which can each be done in $O(m)$ time. We next argue that the repeat-until loop terminates after at most $2b_1 + 2$ iterations. We use that (a) a player-2 edge from $Y^t_\ell = Attr_1(\mathcal{G}^t, T^t_\ell)$ to $V^t \setminus Y^t_\ell$ has to originate from a vertex of T^t_ℓ and (b) if a player-1 attractor contains a vertex set, then it contains also the player-1 attractor of this vertex set. In each iteration we have one of the following situations:

(1) $S^t = \emptyset$: The algorithm terminates.

(2) $Attr_1(\mathcal{G}^t, T_1^t) = V^t$ and $Attr_1(\mathcal{G}^t, T_\ell^t) \neq V^t$ for some $\ell > 1$: We have that $T_1^t \not\subseteq Attr_1(\mathcal{G}^t, T_\ell^t)$ as $T_1^t \subseteq Attr_1(\mathcal{G}^t, T_\ell^t)$ would imply that also $Attr_1(\mathcal{G}^t, T_1^t) = V^t \subseteq Attr_1(\mathcal{G}^t, T_\ell^t)$, which is in contradiction to the assumption. Thus we obtain $|T_1^{t+1}| < |T_1^t|$.

(3) $Attr_1(\mathcal{G}^t, T_1^t) \neq V^t$ and $D^t \cap T_1^t \neq \emptyset$: We immediately get $|T_1^{t+1}| < |T_1^t|$.

(4) $Attr_1(\mathcal{G}^t, T_1^t) \neq V^t$ and $D^t \cap T_1^t = \emptyset$: In this case we have $D^t = Attr_2(\mathcal{G}^t, S^t) = S^t$. Notice that for each vertex $v \in Attr_1(\mathcal{G}^t, T_1^t)$ player 1 has a strategy to reach T_1^t and thus for v to be in D^t, the set D^t has to contain at least one vertex of T_1^t. This implies that in the next iteration $t+1$ we have $T_1^{t+1} = T_1^t$ and $Attr_1(\mathcal{G}^{t+1}, T_1^{t+1}) = Attr_1(\mathcal{G}^t, T_1^t) = V^{t+1}$ and hence either situation (1) or (2).

By the above we have that T_1^t is decreased in at least every second iteration of the repeat-until loop. For $T_1^t = \emptyset$ we have $Attr_1(\mathcal{G}^t, T_1^t) = \emptyset$ and thus $V^{t+1} = \emptyset$, which terminates the algorithm in the next iteration. Thus we have that each iteration takes time $O(km)$ and there are $2b_1 + 2$, i.e., $O(b_1)$, iterations. □

As we can always rearrange the target sets such that $b_1 = \min_{1 \leq \ell \leq k} b_\ell$, this gives an $O(k \cdot \min_{1 \leq \ell \leq k} b_\ell \cdot m)$ time algorithm for generalized Büchi games.

For the final game graph $\mathcal{G}^t$ we have that all vertices are in all the player-1 attractors of the target sets T_ℓ. Thus player 1 can win the game by following one attractor strategy until the corresponding target set is reached and then switching to the attractor strategy of the next target set.

Proposition 7.2.3 (Soundness). *Let V^{t^*} be the set of vertices returned by Algorithm GenBuchiBasic. Each vertex in V^{t^*} is winning for player 1.*

Proof. Let the t^*-th iteration be the last iteration of the algorithm. We have $S^{t^*} = \emptyset$. Thus each vertex of V^{t^*} is contained in $Attr_1(\mathcal{G}^{t^*}, T_\ell^{t^*})$ for each $1 \leq \ell \leq k$. Additionally, either $V^{t^*} = \emptyset$ or $T_\ell^{t^*} \neq \emptyset$ for all $1 \leq \ell \leq k$. Further we have that V^{t^*} is closed for player 2 as only player-2 attractors were removed from V to obtain V^{t^*} (i.e., we apply Lemma 2.5.1 (1) inductively). Hence player 1 has the following winning strategy (with memory) on the vertices of V^{t^*}: On the vertices of $V^{t^*} \setminus \bigcap_{\ell=1}^{k} T_\ell^{t^*}$ first follow the attractor strategy for $T_1^{t^*}$ until a vertex of $T_1^{t^*}$ is reached, then the attractor strategy for $T_2^{t^*}$ until a vertex of $T_2^{t^*}$ is reached and so on until the set $T_k^{t^*}$ is reached; then restart with $T_1^{t^*}$. On the vertices of $\bigcap_{\ell=1}^{k} T_\ell^{t^*} \cap V_1$ player 1 can pick any outgoing edge whose endpoint is in V^{t^*}. Since V^{t^*} is closed for player 2 and $T_\ell^{t^*} \neq \emptyset$ for all $1 \leq \ell \leq k$, this strategy exists, never leaves the set V^{t^*}, and satisfies the generalized Büchi objective. □

For completeness we use that each 1-closed set that avoids some target set is winning for player 2 and that, by Lemma 2.5.1 (3), we can remove such sets from the game graph and recurse on the remaining game graph.

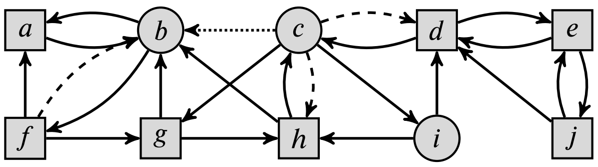

Figure 7.1: Illustration of the hierarchical graph decomposition. Circles denote player 1 vertices, squares denote player 2 vertices. In the original graph G all the edges (solid, dashed, dotted) are present. From the hierarchical graph decomposition we consider the graphs G_1, G_2 and G_3. The graph G_1 contains only the solid edges, the graph G_2 additionally the dashed edges, and G_3 also contains the dotted edge, i.e., $G = G_3$. Let the target sets be $T_1 = \{a, e, i\}$ and $T_2 = \{b, d\}$. Then the set $\{e, j\}$ is a player-2 dominion of G that can be detected in G_1. Its player-2 attractor contains additionally the vertex d. To detect that the remaining vertices are winning for player 1 it is necessary to consider the dotted edge.

Proposition 7.2.4 (Completeness). *Let V^{t^*} be the set returned by Algorithm GenBuchiBasic. Player 2 has a winning strategy from each vertex of $V \setminus V^{t^*}$.*

Proof. By Lemma 2.5.1 (3) it is sufficient to show that, in each iteration t, player 2 has a winning strategy from each vertex of S^t in $\mathcal{G}^t$. Let ℓ be such that $S^t = V^t \setminus Attr_1(\mathcal{G}^t, T^t_\ell)$. By Lemma 2.5.1 (1) the set S^t is closed for player 1 in $\mathcal{G}^t$, that is, player 2 has a strategy that keeps the play within $\mathcal{G}^t[S^t]$ against any strategy of player 1. Since $S^t \cap T^t_\ell = \emptyset$, this strategy is winning for player 2 (i.e., it satisfies coBüchi(T^t_ℓ) and thus the disjunctive co-Büchi objective). □

7.2.2 Our Improved Algorithm

In this subsection we show the following theorem.

Theorem 7.2.5. *Algorithm GenBuchi computes the winning sets for generalized Büchi games with n vertices and k target sets in $O(k \cdot n^2)$ time.*

The $O(k \cdot n^2)$ time Algorithm GenBuchi for generalized Büchi games combines the basic algorithm for generalized Büchi games described above with the method used for the $O(n^2)$ time Büchi game algorithm [CH14], called *hierarchical graph decomposition* [HKW99]. The hierarchical graph decomposition defines for a directed graph $G = (V, E)$ and integers $1 \leq i \leq \lceil \log_2 n \rceil$ the graphs $G_i = (V, E_i)$. Assume the incoming edges of each vertex in G are given in some fixed order in which first the edges from vertices of V_2 and then the edges from vertices of V_1 are listed. The set of edges E_i contains all the outgoing edges of each $v \in V$ with $Outdeg(G, v) \leq 2^i$ and the first 2^i incoming edges of each vertex. Note that $G = G_{\lceil \log_2 n \rceil}$ and $|E_i| \in O(n \cdot 2^i)$. See Figure 7.1 for an example. The running time analysis uses that we can identify small player-2 dominions (i.e., player-1 closed sets that do not intersect at least one of the target sets) that contain $O(2^i)$ vertices

Algorithm GenBuchi: Generalized Büchi games in $O(k \cdot n^2)$ time

```
Input  : game graph $\mathcal{G} = (G, (V_1, V_2))$ with graph $G = (V, E)$ and
         generalized Büchi objective $\bigwedge_{1 \le \ell \le k}$ Büchi$(T_\ell)$
Output : winning set of player 1
$\mathcal{G}^1 \leftarrow \mathcal{G}$
$\{T_\ell^1\} \leftarrow \{T_\ell\}$
$t \leftarrow 0$
repeat
    $t \leftarrow t + 1$
    for $i \leftarrow 1$ to $\lceil \log_2 n \rceil$ do
        construct $G_i^t$
        $Z_i^t \leftarrow \{v \in V_2^t \mid Outdeg(G_i^t, v) = 0\} \cup \{v \in V_1^t \mid Outdeg(G^t, v) > 2^i\}$
        for $1 \le \ell \le k$ do
            $Y_{\ell,i}^t \leftarrow Attr_1(G_i^t, T_\ell^t \cup Z_i^t)$
            $S^t \leftarrow V^t \setminus Y_{\ell,i}^t$
            if $S^t \neq \emptyset$ then player 2 dominion found, continue with line 13
    $D^t \leftarrow Attr_2(G^t, S^t)$
    $\mathcal{G}^{t+1} \leftarrow \mathcal{G}^t \setminus D^t$
    $\{T_\ell^{t+1}\} \leftarrow \{T_\ell^t \setminus D^t\}$
until $D^t = \emptyset$
return $V^t$
```

by considering only G_i. The algorithm first searches for such a set S^t in G_i for $i = 1$ and each target set and then increases i until the search is successful. In this way the time spent for the search is proportional to $k \cdot n$ times the number of vertices in the found dominion, which yields a total running time bound of $O(k \cdot n^2)$. To obtain the $O(k \cdot n^2)$ running time bound, it is crucial to put the loop over the different target sets as the innermost part of the algorithm. Given a game graph $\mathcal{G} = (G, (V_1, V_2))$, we denote by $\mathcal{G}_i$ the game graph where G is replaced by G_i from the hierarchical graph decomposition, i.e., $\mathcal{G}_i = (G_i, (V_1, V_2))$.

Properties of the hierarchical graph decomposition. Lemma 5.3.2, restated below in the notation used in this chapter, describes the essence of why the hierarchical graph decomposition is useful for game graphs with prefix independent objectives. The first part is crucial for correctness: When searching in G_i for a player-1 closed set that does not contain one of the target sets, we can ensure that such a set is also closed for player 1 in G by excluding certain vertices that are missing outgoing edges in G_i from the search. The second part is crucial for the running time argument: Whenever the basic algorithm would remove (i.e., identify as winning for player 2) a set of vertices with at most 2^i vertices, then we can identify this set also by searching in G_i instead of G. The vertices Z_i we exclude for the search on G_i are player-1 vertices with more than 2^i outgoing edges in G and player-2 vertices with no outgoing edges in G_i. Note that the latter can only happen if $Outdeg(G, v) > 2^i$.

Lemma 7.2.6 (Restatement of Lemma 5.3.2). *Let $\mathcal{G} = (G, (V_1, V_2))$ be a game graph with $G = (V, E)$ and $\{G_i\}$ its hierarchical graph decomposition. For $1 \leq i \leq \lceil \log_2 n \rceil$ let Z_i be the set consisting of the player 2 vertices that have no outgoing edge in $\mathcal{G}_i$ and the player 1 vertices with $> 2^i$ outgoing edges in $\mathcal{G}$.*

(1) *If a set $S \subseteq V \setminus Z_i$ is player-1 closed in $\mathcal{G}_i$, then S is player-1 closed in $\mathcal{G}$.*

(2) *If a set $S \subseteq V$ is player-1 closed in $\mathcal{G}$ and $|Attr_2(\mathcal{G}, S)| \leq 2^i$, then (i) $\mathcal{G}_i[S] = \mathcal{G}[S]$, (ii) the set S is in $V \setminus Z_i$, and (iii) S is player-1 closed in $\mathcal{G}_i$.*

For Algorithm GenBuchi this implies that, in all but the last iteration of the repeat-until loop, whenever the graph G_i is considered, a dominion of size at least 2^{i-1} is identified and removed from the graph.

Corollary 7.2.7. *If in Algorithm GenBuchi for some ℓ, t, and $i > 1$ we have that $S^t = V^t \setminus Attr_1(\mathcal{G}_i^t, T_\ell^t \cup Z_i^t)$ is not empty but for $i-1$ the set $V^t \setminus Attr_1(\mathcal{G}_{i-1}^t, T_\ell^t \cup Z_{i-1}^t)$ is empty, then $|Attr_2(\mathcal{G}^t, S^t)| > 2^{i-1}$.*

Proof. By Lemma 2.5.1 (1) S^t is closed for player 1 in $\mathcal{G}_i^t$ and by Lemma 7.2.6 (1) also in $\mathcal{G}^t$. Assume by contradiction that $|Attr_2(\mathcal{G}^t, S^t)| \leq 2^{i-1}$. Then by Lemma 7.2.6 (2) we have that $S^t \subseteq V^t \setminus Z_{i-1}^t$ and S^t is closed for player 1 in $\mathcal{G}_{i-1}^t$. Since this means that in $\mathcal{G}_{i-1}^t$ player 1 has a strategy to keep a play within S^t against any strategy of player 2 and S^t does not contain a vertex of Z_{i-1}^t or T_ℓ^t, the set S^t does not intersect with $Attr_1(\mathcal{G}_{i-1}^t, T_\ell^t \cup Z_{i-1}^t)$, a contradiction to $V^t \setminus Attr_1(\mathcal{G}_{i-1}^t, T_\ell^t \cup Z_{i-1}^t)$ being empty. □

The next two propositions show the correctness of the algorithm by (i) showing that all vertices in the final set V^t are winning for player 1 and (ii) all vertices not in V^t are winning for player 2.

Proposition 7.2.8 (Soundness). *Let V^{t^*} be the set returned by Algorithm GenBuchi. Each vertex in V^{t^*} is winning for player 1.*

Proof. When the algorithm terminates we have $i = \lceil \log_2 n \rceil$ and $S^t = \emptyset$. Since for $i = \lceil \log_2 n \rceil$ we have $G_i^t = G^t$ and $Z_i^t = \emptyset$, the winning strategy of player 1 can be constructed in the same way as for the set returned by Algorithm GenBuchiBasic (cf. Proof of Proposition 7.2.3). □

Proposition 7.2.9 (Completeness). *Let V^{t^*} be the set returned by Algorithm GenBuchi. Player 2 has a winning strategy from each vertex in $V \setminus V^{t^*}$.*

Proof. By Lemma 2.5.1 (3) it is sufficient to show that, in each iteration t, player 2 has a winning strategy in $\mathcal{G}^t$ from each vertex of S^t. For a fixed t with $S^t \neq \emptyset$, let i and ℓ be such that $S^t = V^t \setminus Attr_1(\mathcal{G}_i^t, T_\ell^t \cup Z_i^t)$. By Lemma 2.5.1 (1) the set

S^t is closed for player 1 in $\mathcal{G}_i^t$ and by Lemma 7.2.6 (1) also in $\mathcal{G}^t$. That is, player 2 has a strategy that keeps the play within $\mathcal{G}^t[S^t]$ against any strategy of player 1. Since $S^t \cap T_\ell^t = \emptyset$, this strategy is winning for player 2 (i.e., satisfies the disjunctive co-Büchi objective). □

Finally, the running time of $O(k \cdot n^2)$ follows from Corollary 7.2.7 and the fact that we can construct attractors and the graphs G_i efficiently.

Proposition 7.2.10 (Running time). *Algorithm GenBuchi can be implemented to terminate in $O(k \cdot n^2)$ time.*

Proof. To efficiently construct the graphs G_i^t and the vertex sets Z_i^t we maintain (sorted) lists of the incoming and the outgoing edges of each vertex. These lists can be updated whenever an obsolete entry is encountered in the construction of G_i^t; as each entry is removed at most once, maintaining this data structures takes total time $O(m)$. For a given iteration t of the outer repeat-until loop and the i-th iteration of the inner repeat-until loop we have that the graph G_i^t contains $O(2^i \cdot n)$ edges and both G_i^t and the set Z_i^t can be constructed from the maintained lists in time $O(2^i \cdot n)$. Furthermore, the k attractor computations in the for-loop can be done in time $O(k \cdot 2^i \cdot n)$, thus, for any t, the i-th iteration of the inner repeat-until loop takes time $O(k \cdot 2^i \cdot n)$. The time spent in the iterations up to the i-th iteration forms a geometric series and can thus be bounded by $O(k \cdot 2^i \cdot n)$ as well. When a non-empty set S^t is found in the t-th iteration of the outer repeat-until and in the i-th iteration of the inner repeat-until loop, then by Corollary 7.2.7 we have $|Attr_2(\mathcal{G}^t, S^t)| > 2^{i-1}$. The vertices of $Attr_2(\mathcal{G}^t, S^t)$ are then removed from G^t to obtain G^{t+1} and are not considered further by the algorithm. Thus we can charge the time of $O(k \cdot 2^i \cdot n)$ to identify S^t to the vertices of $Attr_2(\mathcal{G}^t, S^t)$, which yields a bound on the total time spent in the inner repeat-until loop, whenever $S^t \neq \emptyset$, of $O(k \cdot n^2)$. The total time for computing the attractors $Attr_2(\mathcal{G}^t, S^t)$ can be bounded by $O(m)$. Finally, the time for the last iteration of the while-loop, when $S^t = \emptyset$ and $i = \lceil \log_2 n \rceil$, can be bounded by $O(k \cdot 2^{\lceil \log_2 n \rceil} \cdot n) = O(k \cdot n^2)$. □

Remark 7.2.11. *Algorithm GenBuchi can easily be modified to also return winning strategies for both players within the same time bound: For player 2 a winning strategy for the dominion D^t that is identified in iteration t of the algorithm can be constructed by combining his strategy to stay within the set S^t that is closed for player 1 with his attractor strategy to the set S^t. For player 1 we can obtain a winning strategy by combining her attractor strategies to the sets T_ℓ for $1 \leq \ell \leq k$ (as described in the proof of Proposition 7.2.3).*

7.3 Conditional Lower bounds for Generalized Büchi Games

In this section we present two conditional lower bounds, one for dense graphs with $m = \Theta(n^2)$ based on CTC, or equivalently BMM, and one for sparse graphs with

$m = \Theta(n^{1+o(1)})$ based on OVC and thus SETH. The conjectures are stated formally in Section 2.7. We use the existence of memoryless strategies for co-Büchi objectives, see, e.g., [Tho95].

Theorem 7.3.1. *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for generalized Büchi games under Conjecture 2.7.3, i.e., unless CTC & BMM fail. In particular, there is no such algorithm deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

The result can be obtained from a reduction from triangle detection to disjunctive co-Büchi objectives on graphs in Chapter 6, and we present the reduction in terms of game graphs below and illustrate it on an example in Figure 7.2.

Reduction 7.3.2. *Given a graph $G = (V, E)$ (for triangle detection), we build a game graph $\mathcal{G}' = ((V', E'), (V_1, V_2))$ (for generalized Büchi objectives) as follows. As vertices V' we have four copies V^1, V^2, V^3, V^4 of V and a vertex s. A vertex $v^i \in V^i, i \in \{1, 2, 3\}$ has an edge to a vertex $u^{i+1} \in V^{i+1}$ iff $(v, u) \in E$. Moreover, s has an edge to all vertices of V^1, and all vertices of V^4 have an edge to s. All the vertices are owned by player 2, i.e., $V_1 = \emptyset$ and $V_2 = V$. Finally, the generalized Büchi objective is $\bigwedge_{v \in V}$ Büchi(T_v) with $T_v = (V^1 \setminus \{v^1\}) \cup (V^4 \setminus \{v^4\})$.*

The game graph $\mathcal{G}'$ is constructed such that there is a triangle in the graph G if and only if the vertex s is winning for player 2 in the generalized Büchi game on $\mathcal{G}'$. For instance consider the example in Figure 7.2. The graph G has a triangle a,b,c and this triangle gives rise to the following winning strategy for player 2 starting at vertex s. When a play is at the vertex s then player 2 moves to vertex a^1, when at a^1, he moves to b^2, when at b^2 he moves to c^3, when at c^3 he moves to a^4, and finally from a^4, he moves back to s. This strategy does not visit any vertex of the set T_a and thus the conjunctive Büchi objective of player 1 is not satisfied, i.e., player 2 wins. The following lemma shows that also the other direction holds, i.e., that a memoryless winning strategy from s gives rise to a triangle in the original graph. This correspondence between triangles and memoryless winning strategies then gives the correctness of the reduction.

Lemma 7.3.3. *Let $\mathcal{G}'$ be the game graph given by Reduction 7.3.2 for a graph G and let $T_v = (V^1 \setminus \{v^1\}) \cup (V^4 \setminus \{v^4\})$. Then the following statements are equivalent.*

(1) *G contains a triangle.*

(2) *$s \notin W_1(\mathcal{G}', \bigwedge_{v \in V}$ Büchi$(T_v))$.*

(3) *The winning set $W_1(\mathcal{G}', \bigwedge_{v \in V}$ Büchi$(T_v))$ is empty.*

Proof. (1)⇒(2): Assume that G has a triangle with vertices a, b, c and let a^i, b^i, c^i be the copies of a, b, c in V^i. Now a strategy for player 2 in $\mathcal{G}'$ to satisfy coBüchi(T_a) is as follows: When at s, go to a^1; when at a^1, go to b^2; when at b^2, go to c^3; when at c^3, go to a^4; and when at a^4, go to s. As a, b, c form a triangle, all the edges

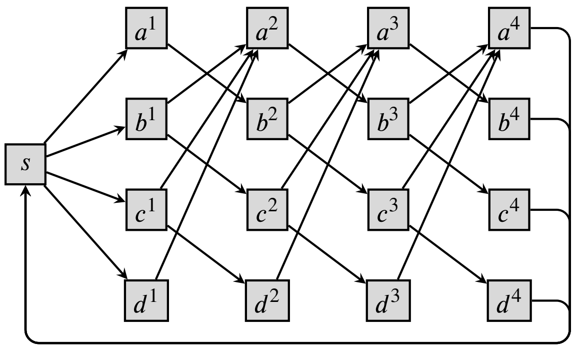

Figure 7.2: Illustration of Reduction 7.3.2, with $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$. The target sets for disjunctive co-Büchi are $T_a = \{b^1, c^1, d^1, b^4, c^4, d^4\}$, $T_b = \{a^1, c^1, d^1, a^4, c^4, d^4\}$, $T_c = \{a^1, b^1, d^1, a^4, b^4, d^4\}$, and $T_d = \{a^1, b^1, c^1, a^4, b^4, c^4\}$.

required by the above strategy exist. When player 2 starts at s and follows the above strategy, then the resulting play forms an infinite path that only uses the vertices s, a^1, b^2, c^3, a^4 and thus satisfies coBüchi(T_a).

(2)⇒(1): Assume that there is a, w.l.o.g. memoryless [Tho95], winning strategy for player 2 starting at s that satisfies coBüchi(T_a). Starting from s, this strategy has to go to a^1, as all other successors of s are contained in T_a and thus the play resulting from the memoryless strategy would violate the coBüchi(T_a) objective. Then the play continues with some vertices $b^2 \in V^2$ and $c^3 \in V^3$ and then, again by the co-Büchi constraint, has to continue with a^4. Now by the construction of $\mathcal{G}'$ we know that there must be edges $(a, b), (b, c), (c, a)$ in the original graph G, i.e. there is a triangle in G.

(2)⇔(3): Notice that when removing s from $\mathcal{G}'$ we obtain an acyclic graph and thus each infinite path has to contain s infinitely often. Thus, if the winning set is non-empty, then there is a cycle winning for some vertex and this cycle is also winning for s. For the converse direction, we have that if s is in the winning set, then the winning set is non-empty. □

The size and the construction time of the game graph $\mathcal{G}'$, given in Reduction 7.3.2, are linear in the size of the original graph G and we have $k = \Theta(n)$ target sets. Thus if we would have a combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$ algorithm for generalized Büchi games, we would immediately get a combinatorial $O(n^{3-\epsilon})$ algorithm for triangle detection, which contradicts CTC (and thus BMM).

Notice that the sets T_v in the above reduction are of linear size but can be reduced to logarithmic size by modifying the graph constructed in Reduction 7.3.2 as follows (more details are given in Section 6.4.1). Remove all edges incident to s and replace them by two complete binary trees. The first tree with s as root and the vertices V^1 as leaves is directed towards the leaves, the second tree with root s and leaves V^4 is directed towards s. Now for each pair v^1, v^4 one can select one vertex per non-root

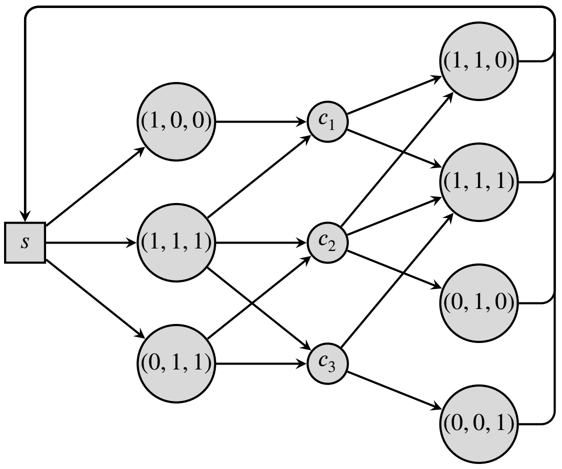

Figure 7.3: Illustration of Reduction 7.3.5, for $S_1 = \{(1,0,0),(1,1,1),(0,1,1)\}$ and $S_2 = \{(1,1,0),(1,1,1),(0,1,0),(0,0,1)\}$.

level of the trees to be in the set T_v such that the only winning path for player 2 starting in s has to use v^1 and each winning path for player 2 to s must pass v^4.

Next we present an $\Omega(m^{2-o(1)})$ lower bound for generalized Büchi objectives in sparse game graphs based on OVC and SETH.

Theorem 7.3.4. *There is no $O(m^{2-\epsilon})$ or $O(\min_{1\leq\ell\leq k} b_\ell \cdot (k \cdot m)^{1-\epsilon})$ time algorithm, for any $\epsilon > 0$, for generalized Büchi games under Conjecture 2.7.5, i.e., unless OVC and SETH fail. In particular, there is no such algorithm for deciding whether the winning set is non-empty or deciding whether a specific vertex is in the winning set.*

The above theorem is by a linear time reduction from OV provided below, and illustrated on an example in Figure 7.3.

Reduction 7.3.5. *Given two sets S_1 and S_2 of d-dimensional vectors, we build the following game graph $\mathcal{G} = ((V,E),(V_1,V_2))$.*

- *The vertices V are given by a start vertex s, sets of vertices S_1 and S_2 representing the sets of vectors, and a set of vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates. The edges E are defined as follows: the start vertex s has an edge to every vertex of S_1 and every vertex of S_2 has an edge to s; further for each $x \in S_1$ there is an edge to $c_i \in \mathcal{C}$ iff $x_i = 1$ and for each $y \in S_2$ there is an edge from $c_i \in \mathcal{C}$ iff $y_i = 1$.*
- *The set of vertices V is partitioned into player 1 vertices $V_1 = S_1 \cup S_2 \cup \mathcal{C}$ and player 2 vertices $V_2 = \{s\}$.*

Finally, the generalized Büchi objective is given by $\bigwedge_{v\in S_2}$ Büchi(T_v) with $T_v = \{v\}$.

The correctness of the reduction is by the following lemma.

Lemma 7.3.6. *Given two sets S_1, S_2 of d-dimensional vectors, the corresponding graph game $\mathcal{G}$ given by Reduction 7.3.5, and $T_v = \{v\}$ for $v \in S_2$, the following statements are equivalent:*

(1) *There exist orthogonal vectors $x \in S_1$ and $y \in S_2$.*

(2) $s \notin W_1(\mathcal{G}, \bigwedge_{v \in S_2} \text{Büchi}(T_v))$

(3) *The winning set $W_1(\mathcal{G}, \bigwedge_{v \in S_2} \text{Büchi}(T_v))$ is empty.*

Proof. W.l.o.g. we assume that the 1-vector, i.e., the vector with all coordinates being 1, is contained in S_2 (adding the 1-vector does not change the result of the OV instance), which guarantees that each vertex $c \in \mathcal{C}$ has at least one outgoing edge. Then a play in the game graph $\mathcal{G}$ proceeds as follows. Starting from s, player 2 chooses a vertex $x \in S_1$; then player 1 first picks a vertex $c \in \mathcal{C}$ and then a vertex $y \in S_2$; then the play goes back to s (at each $y \in S_2$ player 1 has only this choice), starting another cycle of the play.

(1)⇒(2): Assume there are orthogonal vectors $x \in S_1$ and $y \in S_2$. Now player 2 can satisfy coBüchi(T_y) by simply going to x whenever a play is in s. Then player 1 can choose some adjacent $c \in \mathcal{C}$ and then some adjacent vertex in S_2, but as x and y are orthogonal, this c is not connected to y. Thus no play resulting from this strategy visits y.

(2)⇒(1): By $W_1 = V \setminus W_2$ we have that (2) is equivalent to s being in the winning set of player 2 for the conjunctive Büchi game $(\mathcal{G}, \bigwedge_{v \in S_2} \text{Büchi}(T_v))$. Assume $s \in W_2(\mathcal{G}, \bigwedge_{v \in S_2} \text{Büchi}(T_v))$ and consider a corresponding strategy for player 2 that satisfies $\bigvee_{v \in S_2} \text{coBüchi}(T_v)$. Notice that the graph is such that player 2 has to visit at least one of the vertices v in S_1 infinitely often. Moreover, for such a vertex v then player 1 can visit all vertices $v' \in S_2$ that correspond to vectors not orthogonal to v infinitely often. That is, if v has no orthogonal vector, player 1 can satisfy all the Büchi constraints, a contradiction to our assumption that $s \in W_2(\mathcal{G}, \bigwedge_{v \in S_2} \text{Büchi}(T_v))$. Thus there must be a vector $x \in S_1$ such that there exists a vector $y \in S_2$ that is orthogonal to x.

(2) ⇔ (3): Notice that when removing s from the graph we get an acyclic graph and thus each infinite path has to contain s infinitely often. Certainly if s is in the winning set of player 1, this set is non-empty. Thus let us assume there is a vertex v different from s with a winning strategy σ. All (winning) plays starting in v cross s after at most 3 steps and thus the strategy σ is also winning for player 1 when the play starts at s. □

Let $N = \max(|S_1|, |S_2|)$. The number of vertices in the game graph, constructed by Reduction 7.3.5, is $O(N)$, the number of edges m is $\Theta(N \log N)$ (recall that $d \in O(\log N)$), we have $k \in \Theta(N)$ target sets, each of size 1, and the construction can be performed in $O(N \log N)$ time. Thus, if we would have an $O(m^{2-\epsilon})$ or $O(\min_{1 \leq \ell \leq k} b_\ell \cdot (k \cdot m)^{1-\epsilon})$ time algorithm for any $\epsilon > 0$, we would immediately get an $O(N^{2-\epsilon})$ algorithm for OV, which contradicts OVC (and thus SETH).

Remark 7.3.7. *Notice that the conditional lower bounds apply to instances with $k \in \Theta(n^c)$ for arbitrary $0 < c \leq 1$, although the reductions produce graphs with $k \in \Theta(n)$. The instances constructed by the reductions have the property that whenever player 2 has a winning strategy, he also has a winning strategy for a specific co-Büchi target set T_v. Now instead of solving the instance with $\Theta(n)$ many target sets, one can simply consider $\Theta(n^{1-c})$ many instances with $\Theta(n^c)$ target sets and obtain the winning set for player 2 in the original instance by the union of the player 2 winning sets of the new instances. Finally, towards a contradiction, assume there would be an $O((k \cdot f(n,m))^{1-\epsilon})$ time algorithm for $k \in \Theta(n^c)$. Then together we the above observation we would get an $O((k \cdot f(n,m))^{1-\epsilon})$ time algorithm for the original instance.*

Remark 7.3.8. *In both reductions the constructed graph becomes acyclic when deleting the vertex s. Thus, our lower bounds also apply for a broad range of digraph parameters. For instance let w be the DAG-width [Ber+06] of a graph, then there is no $O(f(w) \cdot (k \cdot n^2)^{1-\epsilon})$ time algorithm (under BMM) and no $O(f(w) \cdot (km)^{1-\epsilon})$ time algorithm (under SETH).*

7.4 Generalized Reactivity-1 Games

The input to GR(1) games is a game graph $\mathcal{G} = (G, (V_1, V_2))$, with underlying graph $G = (V, E)$, and subsets of vertices L_ℓ for $1 \leq \ell \leq k_1$ and U_j for $1 \leq j \leq k_2$. The GR(1) objective is the implication between the two generalized Büchi objectives given by $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell)$ and $\bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$, i.e., it is satisfied if either $\bigvee_{\ell=1}^{k_1} \text{coBüchi}(L_\ell)$ or $\bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$ holds. We denote a GR(1) objective by $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell) \rightarrow \bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$. The winning sets of the two players in GR(1) games can be computed in $O(k_1 k_2 \cdot m \cdot n)$ time [JP06] with an extension of the progress measure algorithm of [Jur00] and in $O((k_1 k_2 \cdot n)^{2.5})$ time by combining the reduction to one-pair Streett objectives by [Blo+10] with the algorithm of [CHL15]. In this section we develop an $O(k_1 k_2 \cdot n^{2.5})$ time algorithm by modifying the algorithm of [JP06] to compute dominions instead of winning sets[1]. We further use our $O(k \cdot n^2)$ time algorithm for generalized Büchi games with $k = k_1$ as a subroutine.

Section outline. We first describe a basic, direct algorithm for GR(1) games that is based on repeatedly identifying player-2 dominions in generalized Büchi games. We then show how the progress measure algorithm of [JP06] can be modified to identify player-2 dominions in generalized Büchi games with k_1 Büchi objectives in time proportional to $k_1 \cdot m$ times the size of the dominion. In the $O(k_1 k_2 \cdot n^{2.5})$ time algorithm we use the modified progress measure algorithm in combination with the hierarchical graph decomposition of [CH14] (see also Section 5.3) to identify dominions that contain up to $\sqrt{n}$ vertices and our $O(k_1 \cdot n^2)$ time algorithm for

[1]Note that [JP06] does not contain proofs and that we present a corrected version of a special case of the algorithm of [JP06].

generalized Büchi games to identify dominions with more than $\sqrt{n}$ vertices. Each time we search for a dominion we might have to consider k_2 different subgraphs.

Notation. In the algorithms and their analysis we denote the sets in the t-th iteration of our algorithms with superscript t, in particular $\mathcal{G}^1 = \mathcal{G}$, where $\mathcal{G}$ is the input game graph, G^t is the graph of $\mathcal{G}^t$, V^t is the vertex set of G^t, $V_1^t = V_1 \cap V^t$, $V_2^t = V_2 \cap V^t$, $L_\ell^t = L_\ell \cap V^t$, and $U_j^t = U_j \cap V^t$.

7.4.1 Basic Algorithm for GR(1) Objectives

Similar to generalized Büchi games, the basic algorithm for GR(1) games, described in Algorithm GR(1)Basic, identifies a player-2 dominion S^t, removes the dominion and its player-2 attractor D^t from the graph, and recurses on the remaining game graph $\mathcal{G}^{t+1} = \mathcal{G}^t \setminus D^t$. If no player-2 dominion is found, the remaining set of vertices V^t is returned as the winning set of player 1. Given the set S^t is indeed a player-2 dominion, the correctness of this approach follows from Lemma 2.5.1 (3). A player-2 dominion in $\mathcal{G}^t$ is identified as follows: For each $1 \leq j \leq k_2$ first the player-1 attractor Y_j^t of U_j^t is temporarily removed from the graph. Then a generalized Büchi game with target sets $L_1^t, \dots, L_{k_1}^t$ is solved on $\overline{\mathcal{G}^t \setminus Y_j^t}$. The generalized Büchi player in this game corresponds to player 2 in the GR(1) game and his winning set to a player-2 dominion in the GR(1) game. Note that $V^t \setminus Y_j^t$ is player-1 closed and does not contain U_j^t. Thus if in the game induced by the vertices of $V^t \setminus Y_j^t$ player 2 can win w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell^t)$, then these vertices form a player-2 dominion in the GR(1) game. This observation is formalized in Lemma 7.4.2. Further, we can show that when a player-2 dominion in the GR(1) game on $\mathcal{G}^t$ exists, then for one of the sets U_j^t the winning set of the generalized Büchi game on $\overline{\mathcal{G}^t \setminus Y_j^t}$ is non-empty; otherwise we can construct a winning strategy of player 1 for the GR(1) game on $\mathcal{G}^t$(see Lemma 7.4.3 and Proposition 7.4.4). Note that this algorithm computes a player-2 dominion $O(k_2 \cdot n)$ many times using our $O(k_1 \cdot n^2)$ time generalized Büchi Algorithm GenBuchi from Section 7.2.2. In this subsection we show the following theorem.

Theorem 7.4.1. *Using Theorem 7.2.5, the basic algorithm for GR(1) games computes the winning set of player 1 in $O(k_1 \cdot k_2 \cdot n^3)$ time.*

We first show that the dominions we compute via the generalized Büchi games are indeed player-2 dominions in the GR(1) game.

Lemma 7.4.2. *Let $\mathcal{G}$ be a game graph with a GR(1) objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell) \rightarrow \bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$. Each player-1 dominion D of the game graph $\overline{\mathcal{G}}$ with generalized Büchi objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell)$, for which there is an index $1 \leq j \leq k_2$ with $D \cap U_j = \emptyset$, is a player-2 dominion of $\mathcal{G}$ with the original GR(1) objective.*

Proof. By definition of a dominion, in $\overline{\mathcal{G}}$ player 1 has a strategy that visits all sets L_ℓ infinitely often and only visits vertices of D. But then for some j the target set U_j is not visited at all and thus in $\mathcal{G}$ this strategy is winning for player 2 w.r.t. the GR(1) objective. □

Next we show that each player-2 dominion contains a player-2 dominion that does not intersect with one of the sets U_j, and thus can be computed via generalized Büchi games.

Lemma 7.4.3. *Let $\mathcal{G}$ be a game graph with a GR(1) objective $\bigwedge_{\ell=1}^{k_1}$ Büchi(L_ℓ) $\rightarrow$ $\bigwedge_{j=1}^{k_2}$ Büchi(U_j). Each player-2 dominion D has a subset $D' \subseteq D$ that is a player-2 dominion (in $\mathcal{G}$) with $D' \cap U_j = \emptyset$ for some $1 \leq j \leq k_2$.*

Proof. First, note that by the definition of a 2-dominion we have that on the game graph $\mathcal{G}[D]$ restricted to D player 2 wins from all vertices of D (w.r.t. the GR(1) objective). We show the existence of a 2-dominion $D' \subseteq D$ with $D' \cap U_j = \emptyset$ for some $1 \leq j \leq k_2$ by showing its existence in the game graph $\mathcal{G}[D]$. Since D is closed for player 1, by Lemma 2.6.11 (1) a set $D' \subseteq D$ is a player-2 dominion of $\mathcal{G}$ iff it is a player-2 dominion of $\mathcal{G}[D]$. The proof is by contradiction. We will show that if no such 2-dominion D' with $D' \cap U_j = \emptyset$ for some $1 \leq j \leq k_2$ exists, then player-1 has a winning strategy from all vertices of D in $\mathcal{G}[D]$, a contradiction.

Assume that there does not exist such a player-2 dominion D' in $\mathcal{G}[D]$. We consider for $1 \leq j \leq k_2$ the game graphs $\mathcal{G}^j[D] = \mathcal{G}[D] \setminus Attr_1(\mathcal{G}[D], U_j)$. By Lemma 2.6.11 (2) the vertices of $\mathcal{G}^j[D]$ are 1-closed in $\mathcal{G}[D]$. Thus we have that by Lemma 2.6.11 (1) a 2-dominion in $\mathcal{G}^j[D]$ would be a 2-dominion in $\mathcal{G}[D]$ and thus by our assumption no 2-dominion exists for the GR(1) game on $\mathcal{G}^j[D]$. Hence we have that player 1 wins on all vertices in the GR(1) game on $\mathcal{G}^j[D]$. As $U_j \cap \mathcal{G}^j[D] = \emptyset$, the same strategy is also winning for the disjunctive co-Büchi objective $\bigvee_{\ell=1}^{k_1}$ coBüchi(L_ℓ) on $\mathcal{G}[D]$. Now consider the following strategy of player 1 for the GR(1) game on $\mathcal{G}[D]$. The winning strategy of player 1 on $\mathcal{G}[D]$ is constructed from her winning strategies for the game graphs $\mathcal{G}^j[D]$ and the attractor strategies for $Attr_1(\mathcal{G}[D], U_j)$ for $1 \leq j \leq k_2$ as follows. To win in $\mathcal{G}$, player 1 has to either visit all sets U_j infinitely often or one of the sets L_ℓ only finitely often. Towards visiting all sets U_j, she maintains a counter $c \in \{1, \dots, k_2\}$ that is initialized to 1 and that represents the set U_j she aims at visiting next. As long as the current vertex in a play is contained in $\mathcal{G}^c[D]$, player 1 plays her winning strategy for $\mathcal{G}^c[D]$. If a vertex of $Attr_1(\mathcal{G}[D], U_c)$ is reached, player 1 follows the corresponding attractor strategy until U_c is reached. Then player 1 increases the counter by one or sets the counter to 1 if its value has been k_2 and continues playing the above strategy for the new value of c, and continues like this indefinitely. In each play one of two cases must happen:

- Case 1: After some prefix of the play, the play never reaches $Attr_1(\mathcal{G}[D], U_c)$ for some value of c. Then the play satisfies the disjunctive co-Büchi objective $\bigvee_{\ell=1}^{k_1}$ coBüchi(L_ℓ) and thus the GR(1) objective.

Algorithm GR(1)Basic: GR(1) games in $O(k_1 \cdot k_2 \cdot n^3)$ time

```
Input  : game graph $\mathcal{G} = (G, (V_1, V_2))$ with graph $G = (V, E)$ and
         GR(1) objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell) \to \bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$
Output: winning set of player 1
$\mathcal{G}^1 \leftarrow \mathcal{G}$
$\{U_j^1\} \leftarrow \{U_j\}$; $\{L_\ell^1\} \leftarrow \{L_\ell\}$
$t \leftarrow 0$
repeat
    $t \leftarrow t + 1$
    for $1 \leq j \leq k_2$ do
        $Y_j^t \leftarrow Attr_1(\mathcal{G}^t, U_j^t)$
        $S^t \leftarrow W_1\left(\overline{\mathcal{G}^t \setminus Y_j^t}, \bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell^t \setminus Y_j^t)\right)$
        if $S^t \neq \emptyset$ then break
    $D^t \leftarrow Attr_2(\mathcal{G}^t, S^t)$
    $\mathcal{G}^{t+1} \leftarrow \mathcal{G}^t \setminus D^t$
    $\{U_j^{t+1}\} \leftarrow \{U_j^t \setminus D^t\}$; $\{L_\ell^{t+1}\} \leftarrow \{L_\ell^t \setminus D^t\}$
until $D^t = \emptyset$
return $V^t$
```

- Case 2: For all $c \in \{1, \dots k_2\}$ the set U_c is reached infinitely often. Then the play satisfies the generalized Büchi objective $\bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$ and thus the GR(1) objective.

Hence we have shown that player 1 has a winning strategy from every vertex of D in the GR(1) game on $\mathcal{G}[D]$, a contradiction to D being a 2-dominion in the GR(1) game on $\mathcal{G}$. □

Let the t^*-th iteration be the last iteration of Algorithm GR(1)Basic. For the final game graph $\mathcal{G}^{t^*}$ we can build a winning strategy for player 1 by combining her winning strategies for the disjunctive objective in the subgraphs $\overline{\mathcal{G}_j^{t^*}}$ and the attractor strategies for $Attr_1(\mathcal{G}^{t^*}, U_j)$.

Proposition 7.4.4 (Soundness). *Let V^{t^*} be the set of vertices returned by Algorithm GR(1)Basic. Each vertex in V^{t^*} is winning for player 1.*

Proof. First note that V^{t^*} is closed for player 2 by Lemma 2.5.1 (1). Thus as long as player 1 plays a strategy that stays within V^{t^*}, a play that reaches V^{t^*} will never leave V^{t^*}. The following strategy of player 1 for the vertices of V^{t^*} satisfies this condition. The winning strategy of player 1 is constructed from the winning strategies of the disjunctive co-Büchi player, i.e., player 2, in the generalized Büchi games with game graphs $\overline{\mathcal{G}_j^{t^*}} = \overline{\mathcal{G}^{t^*} \setminus Y_j^{t^*}}$ and objectives $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell^{t^*} \setminus Y_j^{t^*})$ and the attractor strategies for $Attr_1(\mathcal{G}^{t^*}, U_j^{t^*})$ for $1 \leq j \leq k_2$. For a play starting at a vertex of V^{t^*}, player 1 has the following strategy. Player 1 maintains a counter $c \in \{1, \dots, k_2\}$ that is initialized to 1 and proceeds as follows. (1) As long as the current vertex of the play is contained in $G_c^{t^*} = G^{t^*} \setminus Y_c^{t^*}$, player 1 plays her winning strategy for the disjunctive co-Büchi objective on $\mathcal{G}_c^{t^*}$. (2) If a vertex of $Y_c^{t^*} = Attr_1(\mathcal{G}^{t^*}, U_c^{t^*})$ is reached, player 1 follows the corresponding attractor strategy until the play reaches $U_c^{t^*}$. Then player 1 increases the counter by one, or sets the counter to 1 if its value has been k_2, and continues with (1). As the play stays within $V_1^{t^*}$, one of two cases must happen: Case 1: After some prefix of the play for some counter value c the set $Attr_1(\mathcal{G}^{t^*}, U_c^{t^*})$ is never reached, i.e., the play stays within $\mathcal{G}_c^{t^*}$. Then the play satisfies the disjunctive co-Büchi objective $\bigvee_{\ell=1}^{k_1} \text{coBüchi}(L_\ell)$ and thus the GR(1) objective. Case 2: For all $c \in \{1, \dots k_2\}$ the set $U_c^{t^*}$ is reached infinitely often. Then the play satisfies the generalized Büchi objective $\bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$ and thus the GR(1) objective. Hence player 1 wins from all vertices of V^{t^*}. □

We show next that whenever Algorithm GR(1)Basic removes vertices from the game graph, these vertices are indeed winning for player 2. This is due to Lemma 7.4.2 that states that these sets are player-2 dominions in the current game graph, and Lemma 2.5.1 (3) that states that all player-2 dominions of the current game graph $\mathcal{G}^t$ are also winning for player 2 in the original game graph $\mathcal{G}$.

Proposition 7.4.5 (Completeness). *Let V^{t^*} be the set of vertices returned by Algorithm GR(1)Basic. Each vertex in $V \setminus V^{t^*}$ is winning for player 2.*

Proof. By Lemma 2.5.1 (3) it is sufficient to show that in each iteration t with $S^t \neq \emptyset$ player 2 has a winning strategy from the vertices of S^t in $\mathcal{G}^t$. Let j be such that $S^t = W_1\left(\overline{\mathcal{G}^t \setminus Y_j^t}, \bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell^t \setminus Y_j^t)\right)$. We first show that S^t is also a player-1 dominion for the generalized Büchi game on the game graph $\overline{\mathcal{G}^t}$ that includes the vertices of Y_j^t, i.e., that S is a player-2 dominion on $\mathcal{G}^t$. By Lemma 2.5.1 (1) the set $V^t \setminus Y_j^t$ is 1-closed in $\mathcal{G}^t$, i.e., it is 2-closed in $\overline{\mathcal{G}^t}$. Thus by Lemma 2.5.1 (1) each 1-dominion of $\overline{\mathcal{G}^t \setminus Y_j^t}$ is also a 1-dominion in $\overline{\mathcal{G}^t}$. Now as S^t does not contain any vertices of U_j, it is, by Lemma 7.4.2, a player-2 dominion in $\mathcal{G}$ w.r.t. the GR(1) objective. Finally, combining the attractor strategy with the above we have that also $Attr_2(\mathcal{G}^t, S^t)$ is a player-2 dominion in the GR(1) game on $\mathcal{G}$. □

Finally, the running time of Algorithm GR(1)Basic is the product of the number of iterations of the nested loops and the running time for the generalized Büchi algorithm.

Proposition 7.4.6 (Running time). *Algorithm GR(1)Basic runs in $O(k_2 \cdot n \cdot B)$ where B is the running time bound for solving a generalized Büchi game. By Theorem 7.2.5 we have $B = O(k_1 \cdot n^2)$ and thus a total running time of $O(k_1 \cdot k_2 \cdot n^3)$.*

Proof. As in each iteration of the outer loop, except the last one, at least one vertex is removed from the maintained graph, there are at most $O(n)$ iterations. In the inner loop there are k_2 iterations, each with a call to a generalized Büchi game algorithm. Thus, in total we have a running time of $O(k_2 \cdot n \cdot B)$. $\square$

7.4.2 Progress Measure Algorithm for Finding Small Dominions

Our goal for the remaining part of Section 7.4 is to speed up the basic algorithm by computing "small" player-2 dominions faster such that in each iteration of the algorithm a "large" 2-dominion is found and thereby the number of iterations of the algorithm is reduced. To compute small dominions we use a *progress measure* for generalized Büchi games which is a special instance of the more general progress measure for GR(1) games presented in [JP06, Section 3.1], which itself is based on [Jur00]. In this section we first restate the progress measure of [JP06] in our notation and simplified to generalized Büchi, then adapt it to not compute the winning sets but dominions of a given size, and finally give an efficient algorithm to compute the progress measure. The size of a dominion is the number of vertices it contains.

The progress measure of [JP06] is defined as a fixed-point of *ranking functions* that are defined as follows. Let $\bigwedge_{\ell=1}^{k}$ Büchi(T_i) be a generalized Büchi objective. For each $1 \le \ell \le k$ we define the value $\bar{n}_\ell$ to be $\bar{n}_\ell = |V \setminus T_\ell|$. and a *ranking function* $\rho_\ell : V \to \{0, 1, \dots, \bar{n}_\ell, \infty\}$. The intuitive meaning of a value $\rho_\ell(v)$ is the number of moves player 1 needs, when starting from v, to reach a vertex of $T_\ell \cap W_1$, i.e., $\rho_\ell(v)$ will be equal to the rank $rank_1(\mathcal{G}, T_\ell \cap W_1, v)$. As there are only $\bar{n}_\ell$ many vertices which are not in T_ℓ, they can either be reached within $\bar{n}_\ell$ steps or cannot be reached at all. Let $\ell \oplus 1 = \ell + 1$ if $\ell < k$ and $k \oplus 1 = 1$ and analogously $\ell \ominus 1 = \ell - 1$ if $\ell > 1$ and $1 \ominus 1 = k$. The actual value $\rho_\ell(v)$ is defined in a recursive fashion via the values of the successor vertices of v. That is, for $v \notin T_\ell$ we define $\rho_\ell(v)$ by the values $\rho_\ell(w)$ for $(v, w) \in E$. Otherwise, if $v \in T_\ell$, then we already reached T_ℓ and we only have to check whether v is in the winning set. That is, whether v can reach a vertex of the next target set $T_{\ell \oplus 1}$ that is also in the winning set W_1. Hence, for $v \in T_\ell$ we define $\rho_\ell(v)$ by the values $\rho_{\ell \oplus 1}(w)$ for $(v, w) \in E$. For each vertex v we consider all its successor vertices and their values and then pick the minimum if $v \in V_1$ or the maximum if $v \in V_2$. In both cases, $\rho_\ell(v)$ is set to this value increased by 1 for $v \notin T_\ell$. If $v \in T_\ell$, the value is set to ∞ if the minimum (resp. maximum) over the successors is ∞ and to 0 otherwise. This procedure is formalized via two functions. First, best$_\ell(v)$ returns the value of the best neighbor for the player owning v.

$$\mathrm{best}_\ell(v) = \begin{cases} \min_{(v,w)\in E} \rho_{\ell\oplus 1}(w) & \text{if } v \in V_1 \wedge v \in T_\ell\,, \\ \min_{(v,w)\in E} \rho_{\ell}(w) & \text{if } v \in V_1 \wedge v \notin T_\ell\,, \\ \max_{(v,w)\in E} \rho_{\ell\oplus 1}(w) & \text{if } v \in V_2 \wedge v \in T_\ell\,, \\ \max_{(v,w)\in E} \rho_{\ell}(w) & \text{if } v \in V_2 \wedge v \notin T_\ell\,. \end{cases}$$

Second, the function inc_v^ℓ formalizes the increment over the value of the best neighbor that then defines the value of $\rho_\ell(v)$. We define for each set $\{0, 1, \dots, \bar{n}_\ell, \infty\}$ the unary ++ operator as $x{+}{+} = x + 1$ for $x < \bar{n}_\ell$ and $x{+}{+} = \infty$ otherwise.

$$\mathrm{inc}_v^\ell(x) = \begin{cases} 0 & \text{if } v \in T_\ell \wedge x \neq \infty\,, \\ x{+}{+} & \textit{otherwise.} \end{cases}$$

Note that from these definitions it follows a vertex v of T_ℓ can only have $\rho_\ell(v) = 0$ or $\rho_\ell(v) = \infty$. Moreover, we do not have to update $\rho_\ell(v)$ for $v \in T_\ell$ as long as (a) for $v \in V_1$ one w with $(v, w) \in E$ has finite $\rho_{\ell \oplus 1}(w)$ or (b) for $v \in V_2$ all w with $(v, w) \in E$ have finite $\rho_{\ell \oplus 1}(w)$.

The progress measure is now defined as the least simultaneous fixed-point of the operation that updates all ranking functions $\rho_\ell(v)$ to $\max(\rho_\ell(v), \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v)))$. The least fixed-point can be computed via the lifting algorithm (introduced by [Jur00] for parity games) that starts with all the ranking functions $\rho_\ell(.)$ initialized as the zero functions and iteratively updates $\rho_\ell(v)$ to $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$, for all $v \in V$, until the least simultaneous fixed-point is reached.

Given the progress measure, we can decide the generalized Büchi game by the following theorem. Intuitively, player 1 can win starting from a vertex with $\rho_1(v) < \infty$ by keeping a counter ℓ that is initialized to 1, choosing the outgoing edge to $\mathrm{best}_\ell(v)$ whenever at a vertex of V_1, and increasing the counter with $\oplus 1$ when a vertex of T_ℓ is reached.

Theorem 7.4.7. *[JP06, Thm. 1] For a given generalized Büchi game and its progress measure ρ, Player 1 has a winning strategy from a vertex v iff $\rho_1(v) < \infty$.*

As our goal is to compute small dominions, say of size h, instead of the whole winning set, we have to modify the above progress measure as follows. In the definition of the functions ρ_ℓ we redefine the value $\bar{n}_\ell$ to be $\min\{h-1, |V \setminus T_\ell|\}$ instead of $|V \setminus T_\ell|$. The intuition behind this is that if the dominion contains at most h vertices, then from each vertex in the dominion we can reach each set T_ℓ within $h - 1$ steps and we do not care about vertices with a larger distance.

With Algorithm GenBuchiProgressMeasure we give an $O(k \cdot h \cdot m)$ time realization of the lifting algorithm for computing the functions ρ_ℓ. It is a corrected version of the lifting algorithm in [JP06, Section 3.1], tailored to generalized Büchi objectives and dominion computation, and exploits ideas from the lifting algorithm in [EWS05]. We iteratively increase the values $\rho_\ell(v)$ for all pairs (v, ℓ). The main idea for the running time bound is to consider each pair (v, ℓ) at most h times, namely only when the value of $\rho_\ell(v)$ can be increased, and each time we consider a pair, we increase the value of $\rho_\ell(v)$ and only do computations in the order of the degree of v. To this end, we use the following data structure. (1) We maintain a list of pairs (v, ℓ) for which $\rho_\ell(v)$ must be increased because of some update on v's neighbors. (2) We additionally maintain $B_\ell(v)$ for each pair (v, ℓ), which stores the value of $\mathrm{best}_\ell(v)$ from the last time we updated $\rho_\ell(v)$, and (3) a counter $C_\ell(v)$ for $v \in V_1$, which stores the number of successors $w \in \mathit{Out}(v)$ with $\rho_\ell(w) = B_\ell(v)$. Every time $B_\ell(v)$ is updated, we initialize $C_\ell(v)$ using the function $\mathrm{cnt}_\ell(v)$ that counts

Algorithm GenBuchiProgressMeasure: Lifting Algorithm for progress measure of generalized Büchi games

```
Input  : game graph $\mathcal{G} = ((V, E), (V_1, V_2))$,
         generalized Büchi objective $\bigwedge_{1 \le \ell \le k}$ Büchi($T_\ell$), and
         parameter $h \in [1, n] \cap \mathbb{N}$
Output : player 1 dominion / winning set for player 1 if $h = n$
1  foreach $v \in V, 1 \le \ell \le k$ do
2      $B_\ell(v) \leftarrow 0$
3      if $v \in V_1$ then $C_\ell(v) \leftarrow Outdeg(v)$
4      $\rho_\ell(v) \leftarrow 0$
5  $Q \leftarrow \{(v, \ell) \mid v \in V, 1 \le \ell \le k, v \notin T_\ell\}$
6  while $Q \ne \emptyset$ do
7      pick some $(v, \ell) \in Q$ and remove it from $Q$
8      old $\leftarrow \rho_\ell(v)$
9      $B_\ell(v) \leftarrow \mathrm{best}_\ell(v)$
10     if $v \in V_1$ then $C_\ell(v) \leftarrow \mathrm{cnt}_\ell(v)$
11     $\rho_\ell(v) \leftarrow \mathrm{inc}^\ell_v(\mathrm{best}_\ell(v))$
12     foreach $w \in In(v) \setminus T_\ell$ with $(w, \ell) \notin Q$, $\rho_\ell(w) < \infty$ do
13         if $w \in V_1$ and old $= B_\ell(w)$ then
14             $C_\ell(w) \leftarrow C_\ell(w) - 1$
15             if $C_\ell(w) = 0$ then $Q \leftarrow Q \cup \{(w, \ell)\}$
16         else if $w \in V_2$ and $\rho_\ell(v) > B_\ell(w)$ then $Q \leftarrow Q \cup \{(w, \ell)\}$
17     if $\rho_\ell(v) = \infty$ then
18         foreach $w \in In(v) \cap T_{\ell \ominus 1}$ with $(w, \ell \ominus 1) \notin Q$, $\rho_{\ell \ominus 1}(w) < \infty$ do
19             if $w \in V_1$ then
20                 $C_{\ell \ominus 1}(w) \leftarrow C_{\ell \ominus 1}(w) - 1$
21                 if $C_{\ell \ominus 1}(w) = 0$ then $Q \leftarrow Q \cup \{(w, \ell \ominus 1)\}$
22             else if $w \in V_2$ then $Q \leftarrow Q \cup \{(w, \ell \ominus 1)\}$
23 return $\{v \in V \mid \rho_\ell(v) < \infty$ for some $\ell\}$
```

the number of successor vertices w with $\rho_\ell(w) = \mathrm{best}_\ell(v)$. Notice that for $v \in T_\ell$ we only distinguish whether $\rho_{\ell \oplus 1}(v)$ is finite or not.

$$\mathrm{cnt}_\ell(v) = \begin{cases} \left|\{w \in Out(v) \mid \rho_{\ell \oplus 1}(w) < \infty\}\right| & \text{if } v \in T_\ell, \\ \left|\{w \in Out(v) \mid \rho_\ell(w) = \mathrm{best}_\ell(v)\}\right| & \text{if } v \notin T_\ell. \end{cases}$$

Whenever the algorithm considers a pair (v, ℓ), it first computes $\mathrm{best}_\ell(v)$ and $\mathrm{cnt}_\ell(v)$ in $O(Outdeg(v))$ time (l. 9–10), stores these values in $B_\ell(v)$ and $C_\ell(v)$, and updates $\rho_\ell(v)$ to $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(v))$ (l. 11). It then identifies the pairs (w, ℓ), $(w, \ell \ominus 1)$ that are affected by the change of the value $\rho_\ell(v)$ and adds them to the set Q in $O(Indeg(v))$ time (l. 12–22). Thus it spends $O(Outdeg(v) + Indeg(v))$ time whenever $\rho_\ell(v)$ changes, which happens only $O(n)$ times.

In the remainder of the section we prove the following theorem.

Theorem 7.4.8. *For a given generalized Büchi game, Algorithm GenBuchiProgress-Measure is an $O(k \cdot h \cdot m)$ time procedure that either returns a player-1 dominion or the empty set, and, if there is at least one player-1 dominion of size $\leq h$ then it returns a player-1 dominion containing all player-1 dominions of size $\leq h$.*

Remark 7.4.9. *While for the progress measure in [JP06] it holds that $\rho_\ell(v) < \infty$ for some $1 \leq \ell \leq k$ is equivalent to $\rho_{\ell'}(v) < \infty$ for all $1 \leq \ell' \leq k$, this does not hold in general for our modified progress measure ρ. Thus we consider the set $\{v \in V \mid \rho_\ell(v) < \infty$ for some $\ell\}$ as a player-1 dominion and not just the set $\{v \in V \mid \rho_1(v) < \infty\}$.*

The correctness of Algorithm GenBuchiProgressMeasure will be proven by the following invariants that are maintained during the whole algorithm. These invariants show that (a) the data structures Q, B_ℓ, and C_ℓ are maintained correctly, and (b) the values $\rho_\ell(v)$ are bounded from above (i) by $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ and (ii) by the rank $rank_1(\mathcal{G}, T_\ell \cap D, v)$ if v is in a dominion D of size $\leq h$.

Invariant 7.4.10. *The while-loop in Algorithm GenBuchiProgressMeasure has the following loop invariants.*

(1) *For all $v \in V$ and all $1 \leq \ell \leq k$ we have $\rho_\ell(v) \leq \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$.*

(2) *For all $v \in V$ and all $1 \leq \ell \leq k$ we have that if $\rho_\ell(v) \neq 0$ or $v \in T_\ell$, then $\rho_\ell(v) = \mathrm{inc}_v^\ell(B_\ell(v))$.*

(3) *For $v \in V_1$ and all $1 \leq \ell \leq k$ we have*

$$C_\ell(v) = \begin{cases} \left|\{w \in Out(v) \mid \rho_{\ell \oplus 1}(w) < \infty\}\right| & \text{if } v \in T_\ell\,, \\ \left|\{w \in Out(v) \mid \rho_\ell(w) = B_\ell(v)\}\right| & \text{if } v \notin T_\ell, \rho_\ell(v) < \infty\,. \end{cases}$$

(4) *The set Q consists exactly of the pairs (v, ℓ) with $\rho_\ell(v) < \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$.*

(5) *For each player-1 dominion D with $|D| \leq h$ and for each $v \in D$ and all $1 \leq \ell \leq k$ we have $\rho_\ell(v) \leq rank_1(\mathcal{G}, T_\ell \cap D, v) < h$.*

Notice that when the algorithm terminates we have by the Invariants (1) & (4) that $\rho_\ell(v) = \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ for all $v \in V$ and all $1 \leq \ell \leq k$, i.e., the functions $\rho_\ell(v)$ are a fixed-point for the $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ updates. By the following lemmata we prove that the above loop invariants are valid.

Lemma 7.4.11. *Before and after each iteration of the while-loop in Algorithm GenBuchiProgressMeasure we have $\rho_\ell(v) \leq \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ for all $v \in V$ and $1 \leq \ell \leq k$.*

Proof. First note that by the initialization of the functions $\rho_\ell(.)$ with the zero function and by the definition of $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(.))$, the values of $\rho_\ell(v)$ for $v \in V$ never decrease.

We prove the lemma by induction over the iterations of the while-loop. As all $\rho_\ell(v)$ are initialized to 0 and 0 is the minimum value, the inequalities are all satisfied in the *base case* when the algorithm first enters the the while-loop. Now for the *induction step* consider an iteration of the loop and assume the invariant is satisfied before the loop. The value $\rho_\ell(v)$ is only changed when the pair (v, ℓ) is processed and then it is set to $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$. Thus the invariant is satisfied after these iterations. In all the other iterations with different pairs (v', ℓ') the values $\rho_{\ell'}(v')$ are either unchanged or increased . As $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ is monotonic in the values of the neighbors, this can only increase the right side of the inequality and thus this invariant is also satisfied after these iterations. Hence, if the invariant is valid before an iteration of the loop, it is also valid afterwards. □

Lemma 7.4.12. *Before and after each iteration of the while-loop in Algorithm GenBuchiProgressMeasure we have that if $\rho_\ell(v) \neq 0$ or $v \in T_\ell$, then $\rho_\ell(v) = \mathrm{inc}_v^\ell(B_\ell(v))$, for all $v \in V$ and all $1 \leq \ell \leq k$.*

Proof. We prove the lemma by induction over the iterations of the while-loop. As $\rho_\ell(v)$ is initialized to 0, this is trivially satisfied in the *base case*. Now for the *induction step* consider an iteration of the loop and let us assume that the invariant is satisfied before the loop. The values $\rho_\ell(v)$ and $B_\ell(v)$ are only changed when the pair (v, ℓ) is processed and then the invariant is trivially satisfied by the assignments in line 9 and line 11 of the algorithm. □

Lemma 7.4.13. *Before and after each iteration of the while-loop in Algorithm GenBuchiProgressMeasure for $v \in V_1$ we have for all $1 \leq \ell \leq k$*

$$C_\ell(v) = \begin{cases} \left|\{w \in \mathit{Out}(v) \mid \rho_{\ell \oplus 1}(w) < \infty\}\right| & \text{if } v \in T_\ell, \\ \left|\{w \in \mathit{Out}(v) \mid \rho_\ell(w) = B_\ell(v)\}\right| & \text{if } v \notin T_\ell, \rho_\ell(v) < \infty. \end{cases}$$

Proof. We prove the lemma by induction over the iterations of the while-loop. As *base case* consider the point where the algorithm first enters the while-loop. All $\rho_\ell(v)$ and $B_\ell(v)$ are initialized to 0 and thus in both cases the right side of the invariant is equal to $\mathit{Outdeg}(v)$, which is exactly the value assigned to $C_\ell(v)$.

Now for the *induction step* consider an iteration of the loop and let us assume that the Invariants (1)–(3) are satisfied before the loop. Let $v \in V_1$. In an iteration where (v, ℓ) is processed in line 10 we set $C_\ell(v)$ to $\mathrm{cnt}_\ell(v)$ and hence the invariant is satisfied by the definition of $\mathrm{cnt}_\ell(v)$. Otherwise the condition for $C_\ell(v)$ is only affected if a vertex $u \in \mathit{Out}(v)$ is processed. We distinguish the two cases $v \in T_\ell$ and $v \notin T_\ell$.

- If $v \in T_\ell$, then $C_\ell(v)$ is only affected in iterations where pairs $(u, \ell \oplus 1)$ are considered. If the updated value of $\rho_{\ell \oplus 1}(u)$ is less than ∞, then the set $\{w \in \mathit{Out}(v) \mid \rho_{\ell \oplus 1}(w) < \infty\}$ is unchanged and also $C_\ell(v)$ is not changed by the algorithm, i.e., the invariant is still satisfied. Otherwise, if the updated value of $\rho_{\ell \oplus 1}(u)$ is ∞, then u drops out of the set $\{w \in \mathit{Out}(v) \mid \rho_{\ell \oplus 1}(w) < \infty\}$ but also the algorithm decreases $C_\ell(v)$ by one, i.e., again the invariant is satisfied.

- If $v \notin T_\ell$ and $\rho_\ell(v) < \infty$, then $C_\ell(v)$ is only affected in iterations where pairs (u, ℓ) are considered. Let $\rho^o_\ell(u)$ be the value of $\rho_\ell(u)$ before its update. If $\rho^o_\ell(u) > B_\ell(v)$, then $u \notin \{w \in Out(v) \mid \rho_\ell(w) = B_\ell(v)\}$ and thus the set is not affected by the increased value of $\rho_\ell(u)$. In this case the algorithm does not change $C_\ell(v)$ and thus the invariant is satisfied. Otherwise, if $\rho^o_\ell(u) = B_\ell(v)$, then $u \in \{w \in Out(v) \mid \rho_\ell(w) = B_\ell(v)\}$ before the iteration but not after the iteration. In that case the algorithm decreases $C_\ell(v)$ by one and thus the invariant is still satisfied.

  Notice that by Invariants (1) and (2), it cannot happen that $\rho^o_\ell(u) < B_\ell(v)$. To see this, assume by contradiction $\rho^o_\ell(u) < B_\ell(v)$. Let $\text{best}^o_\ell(v)$ denote the value of $\text{best}_\ell(v)$ before the update of $\rho_\ell(u)$. By (1) we have $\rho_\ell(v) \leq \text{inc}^\ell_v(\text{best}^o_\ell(v))$, by the definition of $\text{best}^o_\ell(v)$ and $v \in V_1 \setminus T_\ell$ we have $\text{best}^o_\ell(v) \leq \rho^o_\ell(u)$, and thus by the assumption $\text{best}^o_\ell(v) < B_\ell(v)$. By (2) we have either $\rho_\ell(v) = \text{inc}^\ell_v(B_\ell(v))$ or $\rho_\ell(v) = 0$. In the first case, as $\text{inc}^\ell_v(x)$ is strictly increasing for $x < \infty$, we have $\text{inc}^\ell_v(\text{best}^o_\ell(v)) < \text{inc}^\ell_v(B_\ell(v)) = \rho_\ell(v)$ and thus a contradiction to (1). In the second case the pair (v, ℓ) has not been processed yet and we have a contradiction by $B_\ell(v) = 0$. □

Lemma 7.4.14. *Before and after each iteration of the while-loop in Algorithm GenBuchiProgressMeasure we have that the set Q consists exactly of the pairs (v, ℓ) with $\rho_\ell(v) < \text{inc}^\ell_v(\text{best}_\ell(v))$.*

Proof. The set Q is initialized in line 5 with all pairs (v, ℓ) such that $v \notin T_\ell$. For all of these vertices we have $\text{best}_\ell(v) = 0$ and thus $\text{inc}^\ell_v(\text{best}_\ell(v)) = 1$, i.e., $\rho_\ell(v) = 0 < \text{inc}^\ell_v(\text{best}_\ell(v)) = 1$. Now consider $(v, \ell) \notin Q$, i.e., $v \in T_\ell$. As all $\rho_\ell(v) = 0$, we have $\text{inc}^\ell_v(\text{best}_\ell(v)) = 0$ and thus $\rho_\ell(v) = 0 \not< \text{inc}^\ell_v(\text{best}_\ell(v)) = 0$. Hence, in the *base case* a pair (v, ℓ) is in Q iff $\rho_\ell(v) = 0 < \text{inc}^\ell_v(\text{best}_\ell(v)) = 1$.

Now for the *induction step* consider an iteration of the loop and let us assume that the Invariants (1)–(4) are satisfied before the loop. For the pair (v, ℓ) processed in the iteration, $\rho_\ell(v)$ is set to $\text{inc}^\ell_v(\text{best}_\ell(v))$ and it is removed from Q. Notice that (a) the value of $\rho_\ell(w)$ is only changed when a pair (w, ℓ) processed and (b) $\text{inc}^\ell_v(\text{best}_\ell(w))$ can only increase when other pairs (v, ℓ) are processed. Thus we have to show that in an iteration where the algorithm processes the pair (v, ℓ) all pairs (w, ℓ') with $\rho_{\ell'}(w) = \text{inc}^\ell_v(\text{best}_\ell(w))$ before the iteration and $\rho_{\ell'}(w) < \text{inc}^\ell_v(\text{best}_\ell(w))$ after the iteration are added to the set Q. The only vertices affected by the change of $\rho_\ell(v)$ are those in $In(v)$ which are either (i) not in T_ℓ or (ii) in $T_{\ell \ominus 1}$. In the former case only ρ_ℓ is affected while in the latter case only $\rho_{\ell \ominus 1}$ is affected. Let $\rho^o_\ell(v)$ and $\rho^n_\ell(v)$ be the values before, respectively after, the update of $\rho_\ell(v)$. Notice that if $w \notin T_\ell$ and $\rho_\ell(w) = 0$, then $(w, \ell) \in Q$ by the initialization in line 5. Thus in the following we can assume by Invariant (2) that $\rho_\ell(w) = \text{inc}^\ell_v(B_\ell(w))$ for all $(w, \ell) \notin Q$. We consider the following cases.

- $w \in In(v) \setminus T_\ell$ and $w \in V_1$: Then $\text{inc}^\ell_v(\text{best}_\ell(w)) > \rho_\ell(w)$ iff all $u \in Out(w)$ have $\rho_\ell(u) > B_\ell(w)$. As $(w, \ell) \notin Q$ we know that before the iteration there is at least one $u \in Out(w)$ with $\rho_\ell(u) = B_\ell(w)$. In the case $u \neq v$, $B_\ell(w)$ is not changed during the iteration and thus $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(w)) \not> \rho_\ell(w)$. Hence $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(w)) > \rho_\ell(w)$ iff v is the only vertex in $Out(w)$ with $\rho^o_\ell(v) = B_\ell(w)$. But then, by Invariant (3), $C_\ell(v) = 1$ and thus the algorithm reduces $C_\ell(v)$ to 0 and add (v, ℓ) to the set Q in lines 14–15.

- $w \in In(v) \setminus T_\ell$ and $w \in V_2$: Then $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(w)) > \rho_\ell(w)$ iff there is a vertex $u \in Out(w)$ with $\rho_\ell(u) > B_\ell(w)$. If there would be such an $u \in Out(w)$ different from v then by the induction hypothesis we already have $(v, \ell) \in Q$. Thus we must have that $\rho^n_\ell(v) > B_\ell(w)$ and thus (w, ℓ) is added to Q in line 16 of the algorithm.
- $w \in In(v) \cap T_{\ell \ominus 1}$ and $w \in V_1$: Then $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(w)) > \rho_{\ell\ominus 1}(w)$ iff all $u \in Out(w)$ have $\rho_\ell(u) = \infty$ and $\rho_{\ell\ominus 1}(w) = 0$. This is the case iff v has been the only vertex in $Out(w)$ with $\rho_\ell(v) < \infty$. But then, Invariant (3), $C_\ell(v) = 1$ and thus the algorithm decrements $C_\ell(v)$ to 0 and add $(v, \ell \ominus 1)$ to the set Q in lines 20–21.
- $w \in In(v) \cap T_{\ell\ominus 1}$ and $w \in V_2$: Then, by the definition of inc^ℓ_v, we have $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(w)) > \rho_{\ell\ominus 1}(w)$ iff there is an $u \in Out(w)$ with $\rho_\ell(u) = \infty$ and $\rho_{\ell\ominus 1}(w) = 0$. If there would be such an $u \in Out(w)$ different from v then by the induction hypothesis we already have $(v, \ell \ominus 1) \in Q$. Thus, we have that $\rho^n_\ell(v) = \infty > \rho_{\ell\ominus 1}(w)$ and $\mathrm{inc}^\ell_v(\rho^n_\ell(v)) = \infty > \rho_{\ell\ominus 1}(w) = 0$. In that case $(w, \ell \ominus 1)$ is added to Q in line 22 of the algorithm. □

Lemma 7.4.15. *For each player-1 dominion D with $|D| \leq h$, for each $v \in D$, and all $1 \leq \ell \leq k$ we have $\rho_\ell(v) \leq rank_1(\mathcal{G}, T_\ell \cap D, v) < h$.*

Proof. First notice that as $|D| \leq h$, we have $rank_1(\mathcal{G}, T_\ell \cap D, v) < h$ for all $1 \leq \ell \leq k$ and $v \in D$. Thus we only have to show $\rho_\ell(v) \leq rank_1(\mathcal{G}, T_\ell \cap D, v)$. We show this claim by induction on the number of iterations of the while-loop.

As all functions $\rho_\ell(.)$ are initialized with the 0-function, the invariant is satisfied trivially in the *base case* when the algorithm first enters the while-loop.

Now for the *induction step* consider an iteration of the loop and let us assume all the invariants are satisfied before the loop.

The value $\rho_\ell(v)$ is only updated in line 11 and there it is set to $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(v))$. We distinguish three different cases.

- Assume $v \in V_1$ and $rank_1(\mathcal{G}, T_\ell \cap D, v) = r$ with $1 \leq r < h$. Then, by definition of $rank_1$, there is a $w \in D, w \neq v$, with $(v, w) \in E$ and $rank_1(\mathcal{G}, T_\ell \cap D, w) = r - 1$. Now by the induction assumption, $\rho_\ell(w) \leq r - 1$ at the beginning of the ℓ-th iteration. As $\rho_\ell(w)$ is not changed during the iteration, we have $\rho_\ell(w) \leq r - 1$ and thus $\mathrm{best}_\ell(v) \leq r - 1$. Hence, $\mathrm{inc}^\ell_v(\mathrm{best}_\ell(v)) \leq r$ and the invariant is still satisfied.

- Assume $v \in V_2$ and $rank_1(\mathcal{G}, T_\ell \cap D, v) = r$ with $1 \leq r < h$. Then, by definition of $rank_1$, $rank_1(\mathcal{G}, T_\ell \cap D, w) = r - 1$ for each $(v, w) \in E$ and as D is 2-closed we have $w \in D$. Now by the induction assumption, $\rho_\ell(w) \leq r - 1$ at the beginning of the r-th iteration. As $\rho_\ell(w)$ is not changed during the iteration, we have $\rho_\ell(w) \leq r - 1$ when v is updated for each $(v, w) \in E$ and thus $\mathrm{best}_\ell(v) \leq r - 1$. Hence, $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v)) \leq r$ and the invariant is still satisfied.

- Finally, assume $rank_1(\mathcal{G}, T_\ell \cap D, v) = 0$, that is, $v \in T_\ell$. By the induction hypothesis for all $w \in D$ with $(v, w) \in E$ it holds that $\rho_{\ell \oplus 1}(w) < h$ (and there exists such a $w \in D$) and thus $\mathrm{best}_\ell(v) < h$. Hence, $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v)) = 0$ and the loop invariant is still satisfied.

Thus this loop invariant is maintained during the whole algorithm. □

So far we have shown that the algorithm behaves as described by Invariant 7.4.10. The next lemma provides the ingredients to show that the set $W = \{v \in V \mid \rho_\ell(v) < \infty \text{ for some } \ell\}$ is a player-1 dominion by exploiting the fact that the functions ρ_ℓ form a fixed-point of the update operator.

Lemma 7.4.16. *Let $W = \{v \in V \mid \rho_\ell(v) < \infty$ for some $\ell\}$ be the set computed by Algorithm GenBuchiProgressMeasure.*

(1) *For all $v \in W$ we have that if $\rho_\ell(v) < \infty$, then player 1 has a strategy to reach $\{v' \in T_\ell \mid \rho_\ell(v') = 0\}$ from v by only visiting vertices in W.*

(2) *For all $v \in T_\ell \cap W$ we have that if $\rho_\ell(v) = 0$, then player 1 has a strategy to reach $\{v' \in T_{\ell \oplus 1} \mid \rho_{\ell \oplus 1}(v') = 0\}$ from v by only visiting vertices in W.*

Proof. Recall that by the Invariants (1) and (4) we have $\rho_\ell(v) = \mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ for all $v \in V$ and all $1 \leq \ell \leq k$, i.e., the functions $\rho_\ell(v)$ are a fixed-point of the $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ updates.

(1) Consider a vertex $v \in W$ with $\rho_\ell(v) = d$ for $0 \leq d < h$. We will show by induction in d that then player 1 has a strategy to reach $S = \{v' \in T_\ell \mid \rho_\ell(v') = 0\}$ from v by only visiting vertices in W. For the *base case* we exploit that the functions $\rho_\ell(v)$ are a fixed-point of the $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$ updates. Recall that we assume that each vertex has at least one outgoing edge. By the definition of inc_v^ℓ we have that $\rho_\ell(v) = 0$ only if $v \in T_\ell$ and thus we already have reached S in the base case.

For the *induction step* let us assume the claim holds for all $d' < d$ and consider a vertex v with $\rho_\ell(v) = d$. We distinguish the cases $v \in V_1$ and $v \in V_2$.

- $v \in V_1$: Since ρ is a fixed-point of $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$, we have that there is at least one vertex w with $(v, w) \in E$ and $\rho_\ell(w) = d - 1$. By the induction hypothesis, player 1 has a strategy to reach S starting from w and visiting only vertices in W, and, as player 1 can choose the edge (v, w), also a strategy starting from v.

- $v \in V_2$: Since ρ is a fixed-point of $\mathrm{inc}_v^\ell(\mathrm{best}_\ell(v))$, we have that $\rho_\ell(w) < d$ for all vertices w with $(v, w) \in E$. By the induction hypothesis player 1 has a strategy to reach S starting from any w with $(v, w) \in E$ and visiting only vertices in W, and thus also when starting from v.

Moreover, in both cases only the vertex v is added to the path induced by the strategy, and v is by definition in W. Hence, in both cases player 1 has a strategy to reach S from v by only visiting vertices in W, which concludes the proof of part 1.

(2) Recall that we have $v \in T_\ell$ and $\rho_\ell(v) = 0$. Let $S' = \{v' \in T_{\ell \oplus 1} \mid \rho_{\ell\oplus 1}(v') = 0\}$. Again we distinguish whether $v \in V_1$ or $v \in V_2$.

- If $v \in V_1$, then, as the functions ρ_ℓ form a fixed-point, there is at least one vertex w with $(v, w) \in E$ and $\rho_{\ell\oplus 1}(w) < \infty$. Then by (1) player 1 has a strategy to reach S' starting from w and visiting only vertices in W, and, as player 1 can choose the edge (v, w), also a strategy starting from v.

- If $v \in V_2$, then, as ρ is a fixed-point, we have $\rho_{\ell\oplus 1}(w) < \infty$ for all w with $(v, w) \in E$. Then by (1) player 1 has a strategy to reach S' starting from any w with $(v, w) \in E$ and visiting only vertices in W, and thus also when starting from v.

Again, in both cases only the vertex v is added to the path induced by the strategy, and v is by definition in W. Hence in both cases player 1 has a strategy to reach S' using only vertices of W, which concludes the proof of part 2. □

We now prove the correctness of Algorithm GenBuchiProgressMeasure.

Proposition 7.4.17 (Correctness). *For the game graph $\mathcal{G}$ and the conjunctive Büchi objective $\bigwedge_{1 \le \ell \le k}$ Büchi(T_ℓ), Algorithm GenBuchiProgressMeasure either returns a player-1 dominion or the empty set, and, if exists a player-1 dominion of size $\le h$ then it returns a player-1 dominion containing all player-1 dominions of size $\le h$.*

Proof. We will show that (1) $W = \{v \in V \mid \rho_\ell(v) < \infty \text{ for some } \ell\}$ is a player-1 dominion and that (2) each player-1 dominion of size $\le h$ is contained in W.

(1) The following strategy is winning for player 1 and does not leave W. First, for vertices $v \in W \setminus \bigcup_{\ell=1}^k T_\ell$ pick some ℓ s.t. $\rho_\ell(v) < \infty$ and play the strategy given by Lemma 7.4.16(1) to reach $T_\ell \cap W$. The first time a set T_ℓ is reached, start playing the strategies given by Lemma 7.4.16(2) to first reach the set $T_{\ell\oplus 1} \cap W$, then the set $T_{\ell\oplus 2} \cap W$ and so on. The plays resulting from this strategy visit all target sets infinitely often and never leave the set W. That is, W is a player-1 dominion.

(2) Consider a player-1 dominion D with $|D| \le h$. Then, we have that $rank_1(\mathcal{G}, T_\ell \cap D, v) \le h - 1$ for all T_ℓ and all $v \in D$ and by Invariant (5) that $\rho_\ell(v) \le h - 1$ for all $v \in D$. That is, each $d \in D$ has $\rho_1(v) < \infty$ and thus $D \subseteq W$. □

Finally, we consider the running time of Algorithm GenBuchiProgressMeasure.

Proposition 7.4.18 (Running time). *Algorithm GenBuchiProgressMeasure runs in time $O(k \cdot h \cdot m)$.*

Proof. Notice that the functions $\text{best}_\ell(v)$ and $\text{cnt}_\ell(v)$ can be computed in time proportional to $Outdeg(v)$, while $\text{inc}_v^\ell(.)$ is in constant time. An iteration of the initial foreach loop takes time $O(Outdeg(v))$ and, as each $v \in V$ is considered k times, the entire foreach loop takes time $O(k \cdot m)$. The running time of Algorithm GenBuchiProgressMeasure is dominated by the while-loop. Processing a pair $(v, \ell) \in Q$ takes time $O(Outdeg(v) + Indeg(v))$. Moreover, whenever (v, ℓ) is processed, the value of $\rho_\ell(v)$ is increased by 1 if $v \notin T_\ell$ or by ∞ if $v \in T_\ell$ and thus each pair can be considered at most h times. Hence, for the entire while-loop we have a running time of $O\left(h \cdot \sum_{\ell=1}^{k} \sum_{v \in V}(Outdeg(v) + Indeg(v))\right)$ which can be simplified to $O(k \cdot h \cdot m)$. □

7.4.3 Our Improved Algorithm for GR(1) Games

In this section we present our $O(k_1 k_2 \cdot n^{2.5})$ time algorithm for GR(1) games, the pseudocode is given in Algorithm GR(1) and Procedure kGenBüchiDominion. We prove the following theorem.

Theorem 7.4.19. *Algorithm GR(1) computes the winning sets for GR(1) games with n vertices, k_1 target sets in the antecedent, and k_2 target sets in the consequent in $O(k_1 k_2 \cdot n^{2.5})$ time.*

The overall structure of the algorithm is the same as for the basic algorithm: We search for a player-2 dominion S^t and if one is found, then its player-2 attractor D^t is determined and removed from the current game graph $\mathcal{G}^t$ (with $\mathcal{G}^1 = \mathcal{G}$) to create the game graph for the next iteration, $\mathcal{G}^{t+1}$. If no player-2 dominion exists, then the remaining vertices are returned as the winning set of player 1. The difference to the basic algorithm lies in the way we search for player-2 dominions. Two different procedures are used for this purpose: First we search for "small" dominions with the subroutine kGenBüchiDominion. If no small dominion exists, then we search for player-2 dominions as in the basic algorithm. The guarantee that we find a "large" dominion in the second case (if a player-2 dominion exists) allows us to bound the number of times this can happen. The subroutine kGenBüchiDominion called with parameter h on a game graph $\mathcal{G}$ provides the guarantee to identify all player-2 dominions D for which $|Attr_2(\mathcal{G}, D)| \leq h$, where h is set to $\sqrt{n}$ to achieve the desired running time.

Search for large dominions. If the subroutine kGenBüchiDominion returns an empty set, i.e., when we have for all player-2 dominions D that $|Attr_2(\mathcal{G}^t, D)| > h$, then we search for player-2 dominions as in the basic algorithm: For each $1 \leq j \leq k_2$ first the player-1 attractor Y_j^t of U_j^t is temporarily removed from the graph. Then a generalized Büchi game with target sets $L_1^t \setminus Y_j^t, \ldots, L_{k_1}^t \setminus Y_j^t$ is solved on $\overline{\mathcal{G}^t \setminus Y_j^t}$.

Algorithm GR(1): GR(1) Games in $O(k_1 \cdot k_2 \cdot n^{2.5})$ Time

```
Input  : game graph $\mathcal{G} = (G, (V_1, V_2))$ with graph $G = (V, E)$ and
         GR(1) objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell) \to \bigwedge_{j=1}^{k_2} \text{Büchi}(U_j)$
Output : winning set of player 1
$\mathcal{G}^1 \leftarrow \mathcal{G}$
$\{U_j^1\} \leftarrow \{U_j\}$; $\{L_\ell^1\} \leftarrow \{L_\ell\}$
$t \leftarrow 0$
repeat
    $t \leftarrow t + 1$
    $S^t \leftarrow$ kGenBüchiDominion($\mathcal{G}^t, \{L_\ell^t\}, \{U_j^t\}, \sqrt{n}$)
    if $S^t = \emptyset$ then
        for $1 \le j \le k_2$ do
            $Y_j^t \leftarrow Attr_1(\mathcal{G}^t, U_j^t)$
            $S^t \leftarrow$ GenBüchiGame($\overline{\mathcal{G}^t \setminus Y_j^t}, \bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell^t \setminus Y_j^t)$)
            if $S^t \neq \emptyset$ then break
    $D^t \leftarrow Attr_2(\mathcal{G}^t, S^t)$
    $\mathcal{G}^{t+1} \leftarrow \mathcal{G}^t \setminus D^t$
    $\{U_j^{t+1}\} \leftarrow \{U_j^t \setminus D^t\}$; $\{L_\ell^{t+1}\} \leftarrow \{L_\ell^t \setminus D^t\}$
until $D^t = \emptyset$
return $V^t$
```

The generalized Büchi player in this game corresponds to player 2 in the GR(1) game and his winning set to a player-2 dominion in the GR(1) game, see Lemma 7.4.2.

Procedure kGenBüchiDominion. The procedure searches for player-2 dominions in the GR(1) game, and is guaranteed to return some 2-dominion if there exists a 2-dominion D with $|Attr_2(\mathcal{G}, D)| \le h$, otherwise it might return the empty set. To this end we again consider generalized Büchi games, where the generalized Büchi player corresponds to player 2 in the GR(1) game. We use the same hierarchical graph decomposition as for Algorithm GenBuchi: Let the incoming edges of each vertex be ordered such that the edges from vertices of V_2 come first; for a given game graph $\mathcal{G}^t$ the graph G_i^t contains all vertices of $\mathcal{G}^t$, for each vertex its first 2^i incoming edges, and for each vertex with outdegree at most 2^i all its outgoing edges. The set Z_i^t contains all vertices of V_1^t with outdegree larger than 2^i in G^t and all vertices of V_2^t that have no outgoing edge in G_i^t. We start with $i = 1$ and increase i by one as long as no dominion is found. For a given i we perform the following operations for each $1 \le j \le k_2$: First the player 1 attractor $Y_{i,j}^t$ of $U_j^t \cup Z_i^t$ is determined. Then we search for player-1 dominions on $\overline{\mathcal{G}_i^t \setminus Y_{i,j}^t}$ w.r.t. the objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L_\ell \setminus Y_{i,j}^t)$ with the generalized Büchi progress measure algorithm and parameter $h = 2^i$, i.e., by Theorem 7.4.8 the progress measure algorithm returns all

Procedure kGenBüchiDominion($\mathcal{G}^t$, $\{L^t_\ell\}$, $\{U^t_j\}$, h)

```
Input  : game graph $\mathcal{G}^t = ((V^t, E^t), (V^t_1, V^t_2))$,
         $k_1$ target sets $\{L^t_\ell\}$,
         $k_2$ target sets $\{U^t_j\}$, and
         parameter $h \in [1, n]$
Output : a player-2 dominion in the game graph $\mathcal{G}^t$ with objective
         $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L^t_\ell) \to \bigwedge_{j=1}^{k_2} \text{Büchi}(U^t_j)$ that contains all 2-dominions with at
         most $h$ vertices, or possibly the empty set if no such 2-dominion exists
1 for $i \leftarrow 1$ to $\lceil \log_2(2h) \rceil$ do
2     construct $G^t_i$
3     $Z^t_i \leftarrow \{v \in V^t_2 \mid Outdeg(G^t_i, v) = 0\} \cup \{v \in V^t_1 \mid Outdeg(G^t, v) > 2^i\}$
4     for $1 \le j \le k_2$ do
5         $Y^t_{i,j} \leftarrow Attr_1(\mathcal{G}^t_i, U^t_j \cup Z^t_i)$
6         $X^t_{i,j} \leftarrow$ GenBüchiProgressMeasure($\overline{\mathcal{G}^t_i \setminus Y^t_{i,j}}$, $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L^t_\ell \setminus Y^t_{i,j})$, $2^i$)
7         if $X^t_{i,j} \neq \emptyset$ then return $X^t_{i,j}$
8 return $\emptyset$
```

generalized Büchi dominions in $\overline{\mathcal{G}^t_i \setminus Y^t_{i,j}}$ of size at most h.

The following lemma shows how the properties of the hierarchical graph decomposition extend from generalized Büchi games to GR(1) games. The first part is crucial for correctness: Every non-empty set found by the progress measure algorithm on $\overline{\mathcal{G}^t_i \setminus Y^t_{i,j}}$ for some i and j is indeed a player-2 dominion in the GR(1) game. The second part is crucial for the running time argument: Whenever the basic algorithm for GR(1) games would identify a player-2 dominion D with $|Attr_2(\mathcal{G}, D)| \le 2^i$, then D is also a generalized Büchi dominion in $\overline{\mathcal{G}^t_i \setminus Y^t_{i,j}}$ for some j.

Lemma 7.4.20. *Let the notation be as in Algorithm GR(1).*

(1) *Every $X^t_{i,j} \neq \emptyset$ is a 2-dominion in the GR(1) game on $\mathcal{G}^t$ with $X^t_{i,j} \cap U^t_j = \emptyset$.*

(2) *If in $\mathcal{G}^t$ there exists a player-2 dominion D w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L^t_\ell)$ such that $D \cap U^t_j = \emptyset$ for some $1 \le j \le k_2$ and $|Attr_2(\mathcal{G}^t, D)| \le 2^i$, then D is a player-2 dominion w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L^t_\ell \setminus Y^t_{i,j})$ in $\mathcal{G}^t_i \setminus Y^t_{i,j}$.*

Proof. We prove the two points separately.

(1) By Theorem 7.4.8 the set $X^t_{i,j}$ is a player-2 dominion on $\mathcal{G}^t_i \setminus Y^t_{i,j}$ w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1} \text{Büchi}(L^t_\ell \setminus Y^t_{i,j})$ of player 2. By Lemma 2.5.1 (1) $V^t \setminus Y^t_{i,j}$ is closed for player 1 on $\mathcal{G}^t_i$. Thus by Lemma 2.5.1 (1) $X^t_{i,j}$ is a player-2 dominion w.r.t. the generalized Büchi objective also in $\mathcal{G}^t_i$. As $X^t_{i,j}$ is player-1 closed in $\mathcal{G}^t_i$ and does not intersect with Z^t_i, it is player-1 closed in $\mathcal{G}^t$ by

Lemma 7.2.6 (1). Thus by $E_i^t \subseteq E^t$, the set $X_{i,j}^t$ is a player-2 dominion w.r.t. the generalized Büchi objective also in $\mathcal{G}^t$. Since $X_{i,j}^t$ does not intersect with U_j^t, it is also a player-2 dominion in the GR(1) game on $\mathcal{G}^t$ (cf. Lemma 7.4.2).

(2) Since every player-2 dominion is player-1 closed, we have by Lemma 7.2.6 (2) that (i) $\mathcal{G}^t[D] = \mathcal{G}_i^t[D]$, (ii) D does not intersect with Z_i^t, and (iii) D is player-1 closed in $\mathcal{G}_i^t$. Thus we have that (a) D does not intersect with $Y_{i,j}^t$ and (b) player 2 can play the same winning strategy for the vertices in D on $\mathcal{G}_i^t$ as on $\mathcal{G}^t$. □

From this we can draw the following two corollaries: (1) When we have to go up to i^* in the graph decomposition to find a dominion, then its attractor has size at least 2^{i^*-1} and (2) when kGenBüchiDominion returns an empty set, then all player-2 dominions in the current game graph have more than $h = \sqrt{n}$ vertices.

Corollary 7.4.21. *Let t be some iteration of the repeat-until loop in Algorithm GR(1) and consider the call to* kGenBüchiDominion$(\mathcal{G}^t, \{L_\ell^t\}, \{U_j^t\}, h)$.

(1) *If for some $i > 1$ we have $X_{i,j}^t \neq \emptyset$ but $X_{i-1,j}^t = \emptyset$, then $|Attr_2(\mathcal{G}^t, X_{i,j}^t)| > 2^{i-1}$.*

(2) *If* kGenBüchiDominion$(\mathcal{G}^t, \{L_\ell^t\}, \{U_j^t\}, h)$ *returns the empty set, then for every player-2 dominion D in the GR(1) game we have $|Attr_2(\mathcal{G}^t, D)| > h$.*

Proof. We prove the two points separately.

(1) By Lemma 7.4.20 (1) $X_{i,j}^t$ is a player-2 dominion in the GR(1) game on $\mathcal{G}^t$ with $X_{i,j}^t \cap U_j^t = \emptyset$ and thus in particular a dominion w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1}$ Büchi(L_ℓ^t) such that $X_{i,j}^t \cap U_j^t = \emptyset$. Assume by contradiction $|Attr_2(\mathcal{G}^t, X_{i,j}^t)| \leq 2^{i-1}$. Then by Lemma 7.4.20 (2) we have $X_{i-1,j}^t \neq \emptyset$, a contradiction.

(2) Assume there exists a 2-dominion D with $|Attr_2(\mathcal{G}^t, D)| \leq h$. Then by Lemma 7.4.3 there is also a 2-dominion $D' \subseteq D$ that meets the criteria of Lemma 7.4.20 (2). Let i' be the minimal value such that $|Attr_2(\mathcal{G}^t, D')| \leq 2^{i'}$; we have $i' \leq \lceil \log_2(h) \rceil$. Now, by Lemma 7.4.20 (2), we have that D' is a dominion w.r.t. the generalized Büchi objective $\bigwedge_{\ell=1}^{k_1}$ Büchi$(L_\ell^t \setminus Y_{i',j}^t)$ in $\mathcal{G}_{i'}^t \setminus Y_{i',j}^t$. By the correctness of Algorithm GenBuchiProgressMeasure, the set $X_{i',j}^t$ is a dominion containing D' and thus kGenBüchiDominion$(\mathcal{G}^t = ((V^t, E^t), (V_1^t, V_2^t)), \{L_\ell^t\}, \{U_j^t\}, h)$ returns a non-empty set. □

For the final game graph $\mathcal{G}^{t^*}$ we can build a winning strategy for player 1 in the same way as for Algorithm GR(1)Basic. That is, by combining her winning strategies for the disjunctive objective in the subgraphs $\overline{\mathcal{G}_j^{t^*}}$ and the attractor strategies for $Attr_1(\mathcal{G}^{t^*}, U_j)$.

Proposition 7.4.22 (Soundness). *Let V^{t^*} be the set of vertices returned by Algorithm GR(1). Each vertex in V^{t^*} is winning for player 1.*

Proof. When the algorithm terminates we have $S^{t^*} = \emptyset$. Thus the winning strategy of player 1 can be constructed in the same way as for the set returned by Algorithm GR(1)Basic. (cf. Proof of Proposition 7.4.4) □

Next we show that whenever Algorithm GR(1) removes vertices from the game graph, these vertices are indeed winning for player 2. This is due to Lemma 7.4.20 (1) that states that these sets are 2-dominions in the current game graph and due to Lemma 2.5.1 that states that all player-2 dominions of the current game graph $\mathcal{G}^t$ are also winning for player 2 in the original game graph $\mathcal{G}$.

Proposition 7.4.23 (Completeness). *Let V^{t^*} be the set of vertices returned by Algorithm GR(1). Each vertex in $V \setminus V^{t^*}$ is winning for player 2.*

Proof. By Lemma 2.5.1 (3) it is sufficient to show that in each iteration t with $S^t \neq \emptyset$ player 2 has a winning strategy from the vertices in S^t in $\mathcal{G}^t$. If a non-empty set S^t is returned by kGenBüchiDominion, then S^t is winning for player 2 by Lemma 7.4.20 (1). For the case where S^t is empty after the call to kGenBüchiDominion, the set S^t is determined in the same way as in the basic algorithm for GR(1) games and thus is winning by the correctness of Algorithm GR(1)Basic (cf. Proof of Proposition 7.4.5). □

Finally, as the running time of the subroutine kGenBüchiDominion scales with the size of the smallest player-2 dominion in $\mathcal{G}^t$ and we have only make $O(\sqrt{n})$ many calls to GenBüchiGame, we obtain a running time of $O(k_1 \cdot k_2 \cdot n^{2.5})$.

Proposition 7.4.24 (Running time). *Algorithm GR(1) can be implemented to terminate in $O(k_1 \cdot k_2 \cdot n^{2.5})$ time.*

Proof. We analyze the *total* running time over all iterations of the repeat-until loop. The analysis uses that whenever a player-2 dominion D^t is identified, then the vertices of D^t are removed from the maintained game graph. In particular, we have that whenever kGenBüchiDominion returns an empty set, either at least $h = \sqrt{n}$ vertices are removed from the game graph or the algorithm terminates. Thus this case can happen at most $O(n/h) = O(\sqrt{n})$ times. In this case GenBüchiGame is called k_2 times. By Proposition 7.2.10 this takes total time $O(\sqrt{n} \cdot k_2 \cdot k_1 \cdot n^2) = O(k_1 k_2 \cdot n^{2.5})$.

We next bound the total time spent in kGenBüchiDominion. To efficiently construct the graphs G_i^t and the vertex sets Z_i^t we maintain (sorted) lists of the incoming and the outgoing edges of each vertex. These lists can be updated whenever an obsolete entry is encountered in the construction of G_i^t; as each entry is removed at most once, maintaining these data structures takes total time $O(m)$. Now consider a fixed iteration i of the outer for-loop in kGenBüchiDominion. The graph G_i^t has $O(2^i \cdot n)$ edges and thus, given the above data structure for adjacent edges, the graphs G_i^t and the sets Z_i^t can be constructed in $O(2^i \cdot n)$ time. Further the k_2

attractor computations in the inner for-loop can be done in time $O(k_2 \cdot 2^i \cdot n)$. The running time of iteration i is dominated by the k_2 calls to GenBüchiProgressMeasure. By Theorem 7.4.8 the calls to GenBüchiProgressMeasure in iteration i, with parameter h set to 2^i, take time $O(k_1 k_2 \cdot n \cdot 2^{2i})$. Let i^* be the iteration at which kGenBüchiDominion stops after it is called in the t-th iteration of the repeat-until loop. The running time for this call to kGenBüchiDominion from $i = 1$ to i^* forms a geometric series that is bounded by $O(k_1 k_2 \cdot n \cdot 2^{2i^*})$. By Corollary 7.4.21 either (1) a dominion D with $|Attr_2(\mathcal{G}^t, D)| > 2^{i^*-1}$ vertices is found by kGenBüchiDominion or (2) all dominions in $\mathcal{G}^t$ have more than h vertices. Thus either (2a) a dominion D with more than h vertices is detected in the subsequent call to GenBüchiGame or (2b) there is no dominion in $\mathcal{G}^t$ and t is the last iteration of the algorithm. Case (2b) can happen at most once and its running time is bounded by $O(k_1 k_2 \cdot n \cdot 2^{2\log(h)}) = O(k_1 k_2 \cdot n^2)$. In the cases (1) and (2a) more than 2^{i^*-1} vertices are removed from the graph in this iteration, since $h > 2^{i^*-1}$. We charge to each such vertex $O(k_1 k_2 \cdot n \cdot 2^{i^*}) = O(k_1 k_2 \cdot n \cdot h)$ time. Hence the total running time for these cases is $O(k_1 k_2 \cdot n^2 \cdot h) = O(k_1 k_2 \cdot n^{2.5})$. □

Remark 7.4.25. *Algorithm GR(1) can be modified to additionally return winning strategies for both players. Procedure* GenBüchiProgressMeasure($\mathcal{G}, \phi, h$) *can be modified to return a winning strategy within the returned dominion (as described by Lemma 7.4.16). Procedure* GenBüchiGame *can be modified to return winning strategies for both player in the generalized Büchi game. Thus for player 2 a winning strategy for the dominion D^t that is identified in iteration t of the algorithm can be constructed by combining his winning strategy in the generalized Büchi game in which S^t is identified with his attractor strategy to the set S^t. For player 1 we can obtain a winning strategy in the final iteration of the algorithm by combining for $1 \leq j \leq k_2$ her attractor strategies to the sets U_j with her winning strategies in the generalized Büchi games for each of the game graphs $\overline{\mathcal{G}_i^t \setminus Y_{i,j}^t}$ (as described in the proof of Proposition 7.4.4).*

7.5 Conclusion

In this chapter we consider the algorithmic problem of computing the winning sets for games on graphs with generalized Büchi and GR(1) objectives. We present improved algorithms for both, and conditional lower bounds for generalized Büchi objectives. The existing upper bounds and our conditional lower bounds are tight for (a) for dense graphs, and (b) sparse graphs with constant size target sets. Two interesting open questions are as follows: (1) For sparse graphs with $\theta(n)$ many target sets of size $\theta(n)$ the upper bounds are cubic, whereas the conditional lower bound is quadratic, and closing the gap is an interesting open question. (2) For GR(1) objectives we obtain the conditional lower bounds from generalized Büchi objectives, which are not tight in this case; whether better (conditional) lower bounds can be established also remains open.

CHAPTER 8

Conclusion

This thesis connects two areas of theoretical computer science that are too often treated as isolated from each other: on the one hand algorithm design and computational complexity that are considered to be part of “Track A” of theory, on the other hand model checking, automata, and graph games that belong to “Track B”. By relating the asymptotic worst-case running times of different problems, e.g., the running times of checking the non-emptiness of Rabin automata and solving generalized Büchi games to the running times of CNF-SAT and of combinatorial Boolean matrix multiplication, we connect fundamental algorithmic questions from both areas. From the point of view of “Track B”, we introduce popular conjectures and the concept of fine-grained reductions to obtain conditional lower bounds on running times. Additionally, we highlight the applicability of techniques from graph algorithms to canonical problems in model checking and synthesis. From the point of view of “Track A”, we draw attention to important algorithmic problems in the formal analysis of systems and provide an accessible exposition in the language of graph algorithms. In particular, parity games are one of the rare “natural” problems in NP ∩ coNP for which no polynomial-time algorithm is known and are therefore of high interest for the complexity landscape. Furthermore, symbolic computation is a model of computation that is very relevant in practice and deserves more attention from the theory community; symbolic operations are, at the same time, restricted and powerful and could therefore allow for surprising upper and lower bounds.

We conclude with an overview of the results of this thesis and possible directions for future research that emerge from the different parts of the thesis.

Approximating the minimum cycle mean. We initialize the algorithmic study of approximating mean-payoff objectives on graphs and showed the first running time improvement for dense graphs using fast matrix multiplication. While fast matrix multiplication is a valuable theoretical tool, it is not efficient in practice. Therefore an interesting research direction is to develop other methods for improving the running time and to study the trade-off between the running time and the approximation factor.

Algorithms for MDPs with Streett objectives. For Streett objectives we showed the first sub-quadratic time algorithm for MDPs and improved running times for dense graphs and MDPs. Among others, we applied a sparsification technique and a local graph exploration approach from graph algorithms to obtain faster running times on dense and sparse graphs, respectively. An open question is whether the running time can be improved further or whether there exists a (conditional) lower bound. Insights into either of these directions might also lead to an improved understanding of other algorithmic problems in both graph theory and verification and synthesis, where similar techniques were applied (see Section 4.6).

Parity games. For parity games we showed the first sub-cubic time algorithm for three priorities and an improved set-based symbolic algorithm that is the first with a sub-exponential bound on the number of symbolic steps, and for different parameters has an improved exponential dependence on the number of priorities. In contrast to the exponential number of sets used by the previously best algorithm, we use only a linear number of sets. The major and long-standing theoretical open problem is whether a polynomial-time algorithm for parity games exists. Fine-grained reductions could be the next step in understanding the complexity of parity games. Moreover, symbolic algorithms are of great practical relevance, therefore it would be worth exploring whether some of the algorithmic ideas also lead to practical improvements.

Separating polynomial-time problems with conditional lower bounds. The basis of our conditional lower bounds are our fine-grained reductions from CNF-SAT to disjunctions of reachability and safety objectives on MDPs and from Boolean matrix multiplication to disjunctive safety objectives on graphs and disjunctive reachability queries on MDPs. Based on these, we obtain conditional lower bounds for other objectives on graphs and MDPs and for generalized Büchi games. These results show that algorithmic improvements for fundamental problems in formal verification and reactive synthesis would imply a breakthrough in algorithm design. In combination with further new algorithmic results of this thesis, the conditional lower bounds show for the first time separation results among polynomial-time problems in model checking and synthesis; namely, we show that, under popular assumptions, some problems have a strictly higher worst-case running time for MDPs than for graphs and, similarly, for game graphs than for MDPs or graphs, and we separate the running times of closely related objectives, such as Rabin and Streett objectives. Our results are a first step in analyzing the time complexity of algorithmic problems in formal verification and synthesis beyond complexity classes, and many open questions remain (for some more specific open questions see Sections 6.7 and 7.5). An intriguing research direction would be conditional lower bounds for mean-payoff games that are quantitative generalizations of parity games and have the same rare complexity status; we are not aware of even a slightly super-linear conditional lower bound for mean-payoff games.

Bibliography

[Abb+16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. “Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made”. In: *Symposium on Theory of Computing (STOC).* 2016, pp. 375–388. DOI: 10.1145/2897518.2897653.

[ABV15a] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. “If the Current Clique Algorithms are Optimal, so is Valiant’s Parser”. In: *Symposium on Foundations of Computer Science (FOCS).* 2015, pp. 98–117. DOI: 10.1109/FOCS.2015.16.

[ABV15b] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. “Tight Hardness Results For LCS and other Sequence Similarity Measures”. In: *Symposium on Foundations of Computer Science (FOCS).* 2015, pp. 59–78. DOI: 10.1109/FOCS.2015.14.

[AD94] Rajeev Alur and David L. Dill. “A theory of timed automata”. In: *Theoretical Computer Science* 126 (1994), pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8.

[AF07] Luca de Alfaro and Marco Faella. “An Accelerated Algorithm for 3-Color Parity Games with an Application to Timed Games”. In: *International Conference on Computer Aided Verification (CAV).* 2007, pp. 108–120. DOI: 10.1007/978-3-540-73368-3_13.

[AGM97] Noga Alon, Zvi Galil, and Oded Margalit. “On the Exponent of the All Pairs Shortest Path Problem”. In: *Journal of Computer and System Sciences* 54.2 (1997), pp. 255–262. DOI: 10.1006/jcss.1997.1388. Announced at FOCS’91.

[AH01] Luca de Alfaro and Thomas A. Henzinger. “Interface automata”. In: *International Symposium on Foundations of Software Engineering (FSE).* 2001, pp. 109–120. DOI: 10.1145/503209.503226.

[AH04] Rajeev Alur and Thomas A. Henzinger. “Computer-Aided Verification”. 2004. URL: http://www.cis.upenn.edu/group/cis673/. Unpublished, available at http://www.cis.upenn.edu/group/cis673/.

[AHK02] Rajeev Alur, Thomas. A. Henzinger, and Orna Kupferman. “Alternating-time temporal logic”. In: *Journal of the ACM* 49 (2002), pp. 672–713. DOI: 10.1145/585265.585270.

[AHM01] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. “Symbolic Algorithms for Infinite-State Games”. In: *International Conference on Concurrency Theory (CONCUR).* 2001, pp. 536–550. DOI: 10.1007/3-540-44685-0_36.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AL13] Amir Abboud and Kevin Lewi. "Exact Weight Subgraphs and the k-Sum Conjecture". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 2013, pp. 1–12. DOI: 10.1007/978-3-642-39206-1_1.

[Alf+03] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. "The Element of Surprise in Timed Games". In: *International Conference on Concurrency Theory (CONCUR)*. 2003, pp. 142–156. DOI: 10.1007/978-3-540-45187-7_9.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[AN96] Noga Alon and Moni Naor. "Derandomization, witnesses for Boolean matrix multiplication and construction of perfect hash functions". In: *Algorithmica* 16.4–5 (1996), pp. 434–449. DOI: 10.1007/BF01940874.

[AT04] Rajeev Alur and Salvatore La Torre. "Deterministic generators and games for Ltl fragments". In: *ACM Transactions on Computational Logic* 5.1 (2004), pp. 1–25. DOI: 10.1145/963927.963928.

[AV14] Amir Abboud and Virginia Vassilevska Williams. "Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems". In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 434–443. DOI: 10.1109/FOCS.2014.53.

[AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. "Consequences of Faster Alignment of Sequences". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 2014, pp. 39–51. DOI: 10.1007/978-3-662-43948-7_4.

[AVW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. "Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs". In: *Symposium on Discrete Algorithms (SODA)*. 2016, pp. 377–391. DOI: 10.1137/1.9781611974331.ch28.

[AVY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. "Matching Triangles and Basing Hardness on an Extremely Popular Conjecture". In: *Symposium on Theory of Computing (STOC)*. 2015, pp. 41–50. DOI: 10.1145/2746539.2746594.

[AWY15] Amir Abboud, Richard Ryan Williams, and Huacheng Yu. "More Applications of the Polynomial Method to Algorithm Design". In: *Symposium on Discrete Algorithms (SODA)*. 2015, pp. 218–230. DOI: 10.1137/1.9781611973730.17.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. "Finding and Counting Given Length Cycles". In: *Algorithmica* 17.3 (1997), pp. 209–223. DOI: 10.1007/BF02523189. Announced at ESA'94.

[BC92] Peter Butkovic and Raymond A. Cuninghame-Green. "An $O(n^2)$ algorithm for the maximum cycle mean of an $n \times n$ bivalent matrix." In: *Discrete Applied Mathematics* 35.2 (1992), pp. 157–162. DOI: 10.1016/0166-218X(92)90039-D.

[Bee80] Catriel Beeri. “On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases”. In: *ACM Transactions on Database Systems* 5.3 (1980), pp. 241–259. DOI: 10.1145/320613.320614.

[Ber+06] Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. “DAG-Width and Parity Games”. In: *Symposium on Theoretical Aspects of Computer Science (STACS).* 2006, pp. 524–536. DOI: 10.1007/11672142_43.

[Bey+14] Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. “A constraint-based approach to solving games on infinite graphs”. In: *Symposium on Principles of Programming Languages (POPL).* 2014, pp. 221–234. DOI: 10.1145/2535838.2535860.

[BI15] Arturs Backurs and Piotr Indyk. “Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)”. In: *Symposium on Theory of Computing (STOC).* 2015, pp. 51–58. DOI: 10.1145/2746539.2746612.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[BK15] Karl Bringmann and Marvin Künnemann. “Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping”. In: *Symposium on Foundations of Computer Science (FOCS).* 2015, pp. 79–97. DOI: 10.1109/FOCS.2015.15.

[BKV04] Doron Bustan, Orna Kupferman, and Moshe Y. Vardi. “A Measured Collapse of the Modal μ-Calculus Alternation Hierarchy”. In: *Symposium on Theoretical Aspects of Computer Science (STACS).* 2004, pp. 522–533. DOI: 10.1007/978-3-540-24749-4_46.

[BL69] J. Richard Büchi and Lawrence H. Landweber. “Solving sequential conditions by finite-state strategies”. In: *Transactions of the AMS* 138 (1969), pp. 295–311.

[Blo+07] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. “Interactive presentation: Automatic hardware synthesis from specifications: a case study”. In: *Conference on Design, Automation, and Test in Europe (DATE).* 2007, pp. 1188–1193. URL: http://dl.acm.org/citation.cfm?id=1266366.1266622.

[Blo+09] Roderick Bloem, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. “Synthesizing robust systems”. In: *International Conference on Formal Methods in Computer-Aided Design (FMCAD).* 2009, pp. 85–92. DOI: 10.1109/FMCAD.2009.5351139.

[Blo+10] Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. “Robustness in the Presence of Liveness”. In: *International Conference on Computer Aided Verification (CAV).* 2010, pp. 410–424. DOI: 10.1007/978-3-642-14295-6_36.

[Blo+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. “Synthesis of Reactive(1) designs”. In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 911–938. DOI: 10.1016/j.jcss.2011.08.007.

[Boh+03] Henrik C. Bohnenkamp, Peter van der Stok, Holger Hermanns, and Frits W. Vaandrager. “Cost-Optimization of the IPv4 Zeroconf Protocol”. In: *International Conference on Dependable Systems and Networks (DSN).* 2003, pp. 531–540. DOI: 10.1109/DSN.2003.1209963.

[Bor+11] Endre Boros, Khaled M. Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. "Stochastic Mean Payoff Games: Smoothed Analysis and Approximation Schemes". In: *International Colloquium on Automata, Languages, and Programming (ICALP).* 2011, pp. 147–158. DOI: 10.1007/978-3-642-22006-7_13.

[Bri+11] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. "Faster algorithms for mean-payoff games". In: *Formal Methods in System Design* 38.2 (2011), pp. 97–118. DOI: 10.1007/s10703-010-0105-x.

[Bri14] Karl Bringmann. "Why Walking the Dog Takes Time: Frechet Distance Has No Strongly Subquadratic Algorithms Unless SETH Fails". In: *Symposium on Foundations of Computer Science (FOCS).* 2014, pp. 661–670. DOI: 10.1109/FOCS.2014.76.

[Bro+97] Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. "An Improved Algorithm for the Evaluation of Fixpoint Expressions". In: *Theoretical Computer Science* 178.1-2 (1997), pp. 237–255. DOI: 10.1016/S0304-3975(96)00228-9. Announced at CAV'94.

[Bry86] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8 (1986), pp. 677–691. DOI: 10.1109/TC.1986.1676819.

[BSV03] Henrik Björklund, Sven Sandberg, and Sergei G. Vorobyov. "A Discrete Subexponential Algorithm for Parity Games". In: *Symposium on Theoretical Aspects of Computer Science (STACS).* 2003, pp. 663–674. DOI: 10.1007/3-540-36494-3_58.

[Büc62] J. Richard Büchi. "On a decision method in restricted second-order arithmetic". In: *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960.* 1962, pp. 1–11.

[BV07] Henrik Björklund and Sergei G. Vorobyov. "A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games". In: *Discrete Applied Mathematics* 155.2 (2007), pp. 210–229. DOI: 10.1016/j.dam.2006.04.029.

[CAM13] Krishnendu Chatterjee, Luca de Alfaro, and Rupak Majumdar. "The Complexity of Coverage". In: *International Journal of Foundations of Computer Science* 24.2 (2013), pp. 165–186. DOI: 10.1142/S0129054113400066. Announced at APLAS'08.

[Car+16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. "Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility". In: *Innovations in Theoretical Computer Science (ITCS).* 2016, pp. 261–270. DOI: 10.1145/2840728.2840746.

[CCK12] Krishnendu Chatterjee, Siddhesh Chaubal, and Pritish Kamath. "Faster Algorithms for Alternating Refinement Relations". In: *Conference on Computer Science Logic (CSL).* 2012, pp. 167–182. DOI: 10.4230/LIPIcs.CSL.2012.167.

[CDH10] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. "Qualitative Analysis of Partially-Observable Markov Decision Processes". In: *Symposium on Mathematical Foundations of Computer Science (MFCS)*. 2010, pp. 258–269. DOI: 10.1007/978-3-642-15155-2_24.

[CE81] Edmund M. Clarke and E. Allen Emerson. "Design and synthesis of synchronization skeletons using branching time temporal logic". In: *Logic of Programs*. 1981, pp. 52–71. DOI: 10.1007/BFb0025774.

[Cer+11] Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna, and Rohit Singh. "Quantitative Synthesis for Concurrent Programs". In: *International Conference on Computer Aided Verification (CAV)*. 2011, pp. 243–259. DOI: 10.1007/978-3-642-22110-1_20.

[CG99] Boris V. Cherkassky and Andrew V. Goldberg. "Negative-cycle detection algorithms". In: *Mathematical Programming* 85.2 (1999), pp. 277–311. DOI: 10.1007/s101070050058. Announced at ESA '96.

[CH07] Krishnendu Chatterjee and Thomas A. Henzinger. "Probabilistic Systems with LimSup and LimInf Objectives". In: *International Conference on Infinity in Logic and Computation (ILC)*. 2007, pp. 32–45. DOI: 10.1007/978-3-642-03092-5_4.

[CH11] Krishnendu Chatterjee and Monika Henzinger. "Faster and Dynamic Algorithms For Maximal End-Component Decomposition And Related Graph Problems In Probabilistic Verification". In: *Symposium on Discrete Algorithms (SODA)*. 2011, pp. 1318–1336. DOI: 10.1137/1.9781611973082.101.

[CH12] Krishnendu Chatterjee and Monika Henzinger. "An $O(n^2)$ Time Algorithm for Alternating Büchi Games". In: *Symposium on Discrete Algorithms (SODA)*. 2012, pp. 1386–1399. URL: http://portal.acm.org/citation.cfm?id=2095225&CFID=63838676&CFTOKEN=79617016.

[CH14] Krishnendu Chatterjee and Monika Henzinger. "Efficient and Dynamic Algorithms for Alternating Büchi Games and Maximal End-component Decomposition". In: *Journal of the ACM* 61.3 (2014), p. 15. DOI: 10.1145/2597631. Announced at SODA'11 and SODA'12.

[Cha+13] Krishnendu Chatterjee, Monika Henzinger, Manas Joglekar, and Nisarg Shah. "Symbolic algorithms for qualitative analysis of Markov decision processes with Büchi objectives". In: *Formal Methods in System Design* 42.3 (2013), pp. 301–327. DOI: 10.1007/s10703-012-0180-2. Announced at CAV'11. Implementation available at http://www.cs.cmu.edu/~nkshah/SymbolicMDP.

[Cha+14] Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger, Veronika Loitzenbauer, and Michael A. Raskin. "Approximating the minimum cycle mean". In: *Theoretical Computer Science* 547 (2014), pp. 104–116. DOI: 10.1016/j.tcs.2014.06.031. Announced at GandALF'13.

[Cha+15a] Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia. "Qualitative analysis of POMDPs with temporal logic specifications for robotics applications". In: *International Conference on Robotics and Automation (ICRA)*. 2015, pp. 325–330. DOI: 10.1109/ICRA.2015.7139019.

[Cha+15b] Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Andreas Pavlogiannis, and Prateesh Goyal. "Faster Algorithms for Algebraic Path Properties in Recursive State Machines with Constant Treewidth". In: *Symposium on Principles of Programming Languages (POPL)*. 2015. DOI: 10.1145/2837614.2837624.

[Cha+16a] Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Conditionally Optimal Algorithms for Generalized Büchi Games". In: *Symposium on Mathematical Foundations of Computer Science (MFCS)*. 2016, 25:1–25:15. DOI: 10.4230/LIPIcs.MFCS.2016.25. Full version available at http://arxiv.org/abs/1607.05850.

[Cha+16b] Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Improved Set-Based Symbolic Algorithms for Parity Games". 2016. Unpublished.

[Cha+16c] Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. "Model and Objective Separation with Conditional Lower Bounds: Disjunction is Harder than Conjunction". In: *Symposium on Logic in Computer Science (LICS)*. 2016, pp. 197–206. DOI: 10.1145/2933575.2935304. Full version available at http://arxiv.org/abs/1602.02670.

[Cha+16d] Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. "Algorithms for algebraic path properties in concurrent systems of constant treewidth components". In: *Symposium on Principles of Programming Languages (POPL)*. 2016, pp. 733–747. DOI: 10.1145/2837614.2837624.

[Cha07] Krishnendu Chatterjee. "Stochastic ω-Regular Games". PhD thesis. University of California at Berkeley, 2007.

[Cha10] Timothy M. Chan. "More Algorithms for All-Pairs Shortest Paths in Weighted Graphs". In: *SIAM Journal on Computing* 39.5 (2010), pp. 2075–2089. DOI: 10.1137/08071990X. Announced at STOC'07.

[Che+17] Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Veronika Loitzenbauer, and Nikos Parotsidis. "Faster Algorithms for Computing Maximal 2-Connected Subgraphs in Sparse Directed Graphs". In: *Symposium on Discrete Algorithms (SODA)*. 2017. To appear.

[CHL15] Krishnendu Chatterjee, Monika Henzinger, and Veronika Loitzenbauer. "Improved Algorithms for One-Pair and k-Pair Streett Objectives". In: *Symposium on Logic in Computer Science (LICS)*. 2015, pp. 269–280. DOI: 10.1109/LICS.2015.34. Full version available at http://arxiv.org/abs/1410.0833.

[CHP07] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. "Generalized Parity Games". In: *International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*. Vol. 4423. 2007, pp. 153–167. DOI: 10.1007/978-3-540-71389-0_12.

[CHP11] Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. "Timed Parity Games: Complexity and Robustness". In: *Logical Methods in Computer Science* 7.4 (2011). DOI: 10.2168/LMCS-7(4:8)2011.

[Chu62] Alonzo Church. "Logic, arithmetic, and automata". In: *Proceedings of the International Congress of Mathematicians*. Institut Mittag-Leffler, 1962, pp. 23–35.

[Cim+00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. "NUSMV: A New Symbolic Model Checker". In: *International Journal on Software Tools for Technology Transfer (STTT)* 2.4 (2000), pp. 410–425. DOI: 10.1007/s100090050046.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. "The Complexity of Satisfiability of Small Depth Circuits". In: *International Workshop on Parametrized and Exact Computation (IWPEC), Revised Selected Papers.* 2009, pp. 75–85. DOI: 10.1007/978-3-642-11269-0_6.

[CJH03] Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. "Simple stochastic parity games". In: *Conference on Computer Science Logic (CSL).* 2003, pp. 100–113. DOI: 10.1007/978-3-540-45220-1_11.

[Com08] The Association for Computing Machinery. *ACM Turing Award Honors Founders of Automatic Verification Technology.* 2008. URL: http://www.acm.org/press-room/awards/turing-award-07/ (visited on 10/17/2016).

[CP13] Krishnendu Chatterjee and Vinayak S. Prabhu. "Synthesis of memory-efficient, clock-memory free, and non-Zeno safety controllers for timed systems". In: *Information and Computation* 228 (2013), pp. 83–119. DOI: 10.1016/j.ic.2013.04.003.

[CW96] Edmund M. Clarke and Jeannette M. Wing. "Formal Methods: State of the Art and Future Directions". In: *ACM Computing Surveys* 28.4 (1996), pp. 626–643. DOI: 10.1145/242223.242257.

[CY95] Costas Courcoubetis and Mihalis Yannakakis. "The Complexity of Probabilistic Verification". In: *Journal of the ACM* 42.4 (1995), pp. 857–907. DOI: 10.1145/210332.210339.

[DG98] Ali Dasdan and Rajesh K. Gupta. "Faster Maximum and Minimum Mean Cycle Algorithms for System-Performance Analysis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17.10 (1998), pp. 889–899. DOI: 10.1109/43.728912.

[DH09] Evgeny Dantsin and Edward A. Hirsch. "Worst-Case Upper Bounds". In: *Handbook of Satisfiability.* Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 403–424. DOI: 10.3233/978-1-58603-929-5-403.

[Dil89] David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits.* The MIT Press, 1989.

[DM10] Manfred Droste and Ingmar Meinecke. "Describing Average- and Longtime-Behavior by Weighted MSO Logics". In: *Symposium on Mathematical Foundations of Computer Science (MFCS).* 2010, pp. 537–548. DOI: 10.1007/978-3-642-15155-2_47.

[DPC09] Alexandre Duret-Lutz, Denis Poitrenaud, and Jean-Michel Couvreur. "On-the-fly Emptiness Check of Transition-Based Streett Automata". In: *International Symposium on Automated Technology for Verification and Analysis (ATVA).* 2009, pp. 213–227. DOI: 10.1007/978-3-642-04761-9_17.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. "The Complexity of Tree Automata and Logics of Programs (Extended Abstract)". In: *Symposium on Foundations of Computer Science (FOCS).* 1988, pp. 328–337. DOI: 10.1109/SFCS.1988.21949.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. "Tree Automata, Mu-Calculus and Determinacy (Extended Abstract)". In: *Symposium on Foundations of Computer Science (FOCS).* 1991, pp. 368–377. DOI: 10.1109/SFCS.1991.185392.

[EJ99] E. Allen Emerson and Charanjit S. Jutla. "The Complexity of Tree Automata and Logics of Programs". In: *SIAM Journal on Computing* 29.1 (1999), pp. 132–158. DOI: 10.1137/S0097539793304741.

[EL86] E. Allen Emerson and Chin-Laung Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract)". In: *Symposium on Logic in Computer Science (LICS).* 1986, pp. 267–278.

[EM79] Andrzej Ehrenfeucht and Jan Mycielski. "Positional strategies for mean payoff games". In: *International Journal of Game Theory* 8.2 (1979), pp. 109–113. DOI: 10.1007/BF01768705.

[EWS05] Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. "Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata". In: *SIAM Journal on Computing* 34.5 (2005), pp. 1159–1175. DOI: 10.1137/S0097539703420675.

[FH10] Nathanaël Fijalkow and Florian Horn. "The surprizing complexity of reachability games". In: *CoRR* abs/1010.2420 (2010). URL: http://arxiv.org/abs/1010.2420.

[FKP05] Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. "Temporal Logic Motion Planning for Mobile Robots". In: *International Conference on Robotics and Automation (ICRA).* 2005, pp. 2020–2025. DOI: 10.1109/ROBOT.2005.1570410.

[Fra86] Nissim Francez. *Fairness.* New York: Springer, 1986. DOI: 10.1007/978-1-4612-4886-6.

[Fre76] Michael L. Fredman. "New Bounds on the Complexity of the Shortest Path Problem". In: *SIAM Journal on Computing* 5.1 (1976), pp. 83–89. DOI: 10.1137/0205006.

[Fri09] Oliver Friedmann. "An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it". In: *Symposium on Logic in Computer Science (LICS).* 2009, pp. 145–156. DOI: 10.1109/LICS.2009.27.

[FV97] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, 1997. DOI: 10.1007/978-1-4612-4054-9.

[GCH13] Yashdeep Godhal, Krishnendu Chatterjee, and Thomas A. Henzinger. "Synthesis of AMBA AHB from formal specification: A case study". In: *Journal of Software Tools Technology Transfer* 15.5-6 (2013), pp. 585–601. DOI: 10.1007/s10009-011-0207-9.

[Gen14] Raffaella Gentilini. "A note on the approximation of mean-payoff games". In: *Information Processing Letters* 114.7 (2014), pp. 382–386. DOI: 10.1016/j.ipl.2014.02.010. Announced at CILC'11.

[GKK88] V.A. Gurvich, A.V. Karzanov, and L.G. Khachiyan. "Cyclic games and an algorithm to find minimax cycle means in directed graphs". In: *USSR Computational Mathematics and Mathematical Physics* 28.5 (1988), pp. 85–91. DOI: 10.1016/0041-5553(88)90012-2.

[GO12] Anka Gajentaan and Mark H. Overmars. "On a class of O(n^2) problems in computational geometry". In: *Computational Geometry* 45.4 (2012), pp. 140–152. DOI: 10.1016/j.comgeo.2011.11.006.

[Gol95] Andrew V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM Journal on Computing* 24.3 (1995), pp. 494–504. DOI: 10.1137/S0097539792231179.

[Hen+15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture". In: *Symposium on Theory of Computing (STOC)*. 2015, pp. 21–30. DOI: 10.1145/2746539.2746609.

[Hen95] Thomas A. Henzinger. "Hybrid automata with finite bisimulations". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 1995, pp. 324–335. DOI: 10.1007/3-540-60084-1_85.

[HKL15] Monika Henzinger, Sebastian Krinninger, and Veronika Loitzenbauer. "Finding 2-Edge and 2-Vertex Strongly Connected Components in Quadratic Time". In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 2015, pp. 713–724. DOI: 10.1007/978-3-662-47672-7_58.

[HKR02] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. "Fair Simulation". In: *Information and Computation* 173.1 (2002), pp. 64–81. DOI: 10.1006/inco.2001.3085.

[HKW99] Monika Henzinger, Valerie King, and Tandy Warnow. "Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology". In: *Algorithmica* 24.1 (1999), pp. 1–13. DOI: 10.1007/PL00009268. Announced at SODA'96.

[HMR05] Thomas A. Henzinger, Rupak Majumdar, and Jean-François Raskin. "A Classification of Symbolic Transition Systems". In: *ACM Transactions on Computational Logic* 6.1 (2005), pp. 1–32. DOI: 10.1145/1042038.1042039.

[HO93] Mark Hartmann and James B. Orlin. "Finding Minimum Cost to Time Ratio Cycles With Small Integral Transit Times". In: *Networks* 23.6 (1993), pp. 567–574. DOI: 10.1002/net.3230230607.

[Hol93] Gerard J. Holzmann. "Design and Validation of Protocols: A Tutorial". In: *Computer Networks and ISDN Systems* 25.9 (1993), pp. 981–1017. DOI: 10.1016/0169-7552(93)90095-L.

[Hol97] Gerard J. Holzmann. "The Model Checker SPIN". In: *IEEE Transactions on Software Engineering* 23.5 (1997), pp. 279–295. DOI: 10.1109/32.588521.

[How60] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[HT96] Monika Henzinger and Jan Arne Telle. "Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning". In: *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*. 1996, pp. 16–27. DOI: 10.1007/3-540-61422-2_117.

[HZ10] Thomas Dueholm Hansen and Uri Zwick. "Lower Bounds for Howard's Algorithm for Finding Minimum Mean-Cost Cycles". In: *International Symposium on Algorithms and Computation (ISAAC).* 2010, pp. 415–426. DOI: 10.1007/978-3-642-17517-6_37.

[Imm81] Neil Immerman. "Number of quantifiers is better than number of tape cells". In: *Journal of Computer and System Sciences* (1981), pp. 384–406. DOI: 10.1016/0022-0000(81)90039-8.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. "Which Problems Have Strongly Exponential Complexity?" In: *Journal of Computer and System Sciences* 63.4 (2001), pp. 512–530. DOI: 10.1006/jcss.2001.1774.

[JGB05] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem. "Program Repair as a Game". In: *International Conference on Computer Aided Verification (CAV).* 2005, pp. 226–238. DOI: 10.1007/11513988_23.

[JP06] Sudeep Juvekar and Nir Piterman. "Minimizing Generalized Büchi Automata". In: *International Conference on Computer Aided Verification (CAV).* 2006, pp. 45–58. DOI: 10.1007/11817963_7.

[JPZ08] Marcin Jurdziński, Mike Paterson, and Uri Zwick. "A Deterministic Subexponential Algorithm for Solving Parity Games". In: *SIAM Journal on Computing* 38.4 (2008), pp. 1519–1532. DOI: 10.1137/070686652.

[Jr78] Sheldon B. Akers Jr. "Binary Decision Diagrams". In: *IEEE Transactions on Computers* C-27.6 (1978), pp. 509–516. DOI: 10.1109/TC.1978.1675141.

[Jur00] Marcin Jurdziński. "Small Progress Measures for Solving Parity Games". In: *Symposium on Theoretical Aspects of Computer Science (STACS).* 2000, pp. 290–301. DOI: 10.1007/3-540-46541-3_24.

[Jur98] Marcin Jurdziński. "Deciding the Winner in Parity Games is in UP ∩ co-UP". In: *Information Processing Letters* 68.3 (1998), pp. 119–124. DOI: 10.1016/S0020-0190(98)00150-1.

[JV13] Zahra Jafargholi and Emanuele Viola. "3SUM, 3XOR, Triangles". In: *Electronic Colloquium on Computational Complexity (ECCC)* 20 (2013), p. 9. URL: http://eccc.hpi-web.de/report/2013/009.

[Kar78] Richard M. Karp. "A characterization of the minimum cycle mean in a digraph". In: *Discrete Mathematics* 23.3 (1978), pp. 309–311. DOI: 10.1016/0012-365X(78)90011-0.

[Kav12] Telikepalli Kavitha. "Faster Algorithms for All-Pairs Small Stretch Distances in Weighted Graphs". In: *Algorithmica* 63.1-2 (2012), pp. 224–245. DOI: 10.1007/s00453-011-9529-y. Announced at FSTTCS'07.

[Kel76] Robert M. Keller. "Formal Verification of Parallel Programs". In: *Communications of the ACM* 19.7 (1976), pp. 371–384. DOI: 10.1145/360248.360251.

[Kla02] Hartmut Klauck. "Algorithms for Parity Games". In: *Automata Logics, and Infinite Games: A Guide to Current Research.* Ed. by Erich Grädel, Wolfgang Thomas, and Thomas Wilke. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 107–129. ISBN: 978-3-540-36387-3. DOI: 10.1007/3-540-36387-4_7.

[KNP00] Marta Kwiatkowska, Gethin Norman, and David Parker. "Verifying Randomized Distributed Algorithms with PRISM". In: *Workshop on Advances in Verification (WAVE).* 2000.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. "PRISM 4.0: Verification of Probabilistic Real-Time Systems". In: *International Conference on Computer Aided Verification (CAV).* 2011, pp. 585–591. DOI: 10.1007/978-3-642-22110-1_47.

[KO81] Richard M. Karp and James B. Orlin. "Parametric shortest path algorithms with an application to cyclic staffing". In: *Discrete Applied Mathematics* 3.1 (1981), pp. 37–45. DOI: 10.1016/0166-218X(81)90026-3.

[Kop96] Peter W. Kopke. "The Theory of Rectangular Hybrid Automata". PhD thesis. Cornell University, 1996.

[Koz83] Dexter Kozen. "Results on the propositional μ-calculus". In: *Theoretical Computer Science* 27.3 (1983), pp. 333–354. DOI: 10.1016/0304-3975(82)90125-6.

[KP09] Wouter Kuijper and Jaco van de Pol. "Computing Weakest Strategies for Safety Games of Imperfect Information". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS).* 2009, pp. 92–106. DOI: 10.1007/978-3-642-00768-2_10.

[KPB94] Sriram C. Krishnan, Anuj Puri, and Robert K. Brayton. "Deterministic Ω Automata vis-a-vis Deterministic Buchi Automata". In: *International Symposium on Algorithms and Computation (ISAAC).* 1994, pp. 378–386. DOI: 10.1007/3-540-58325-4_202.

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. "Higher Lower Bounds from the 3SUM Conjecture". In: *Symposium on Discrete Algorithms (SODA).* 2016, pp. 1272–1287. DOI: 10.1137/1.9781611974331.ch89.

[KV05] Orna Kupferman and Moshe Y. Vardi. "From linear time to branching time". In: *ACM Transactions on Computational Logic* 6.2 (2005), pp. 273–294. DOI: 10.1145/1055686.1055689.

[KV98] Orna Kupferman and Moshe Y. Vardi. "Freedom, Weakness, and Determinism: From Linear-Time to Branching-Time". In: *Symposium on Logic in Computer Science (LICS).* 1998, pp. 81–92. DOI: 10.1109/LICS.1998.705645.

[Law76] Eugène L. Lawler. *Combinatorial optimization: Networks and Matroids.* Dover Publications, 1976.

[Le 14] François Le Gall. "Powers of Tensors and Fast Matrix Multiplication". In: *International Symposium on Symbolic and Algebraic Computation (ISSAC).* 2014, pp. 296–303. DOI: 10.1145/2608628.2627493.

[Lee02] Lillian Lee. "Fast Context-free Grammar Parsing Requires Fast Boolean Matrix Multiplication". In: *Journal of the ACM* 49.1 (2002), pp. 1–15. DOI: 10.1145/505241.505242.

[Lee59] C. Y. Lee. "Representation of Switching Circuits by Binary-Decision Programs". In: *Bell System Technical Journal* 38.4 (1959), pp. 985–999. DOI: 10.1002/j.1538-7305.1959.tb01585.x.

[LH00] Timo Latvala and Keijo Heljanko. “Coping With Strong Fairness”. In: *Fundamenta Informaticae* 43.1-4 (2000), pp. 175–193. DOI: 10.3233/FI-2000-43123409.

[LW17] Kasper Green Larsen and Ryan Williams. “Faster Online Matrix-Vector Multiplication”. In: *Symposium on Discrete Algorithms (SODA).* 2017. To appear, available at http://arxiv.org/abs/1605.01695.

[Mad02] Omid Madani. “Polynomial Value Iteration Algorithms for Deterministic MDPs”. In: *Conference in Uncertainty in Artificial Intelligence (UAI).* 2002, pp. 311–318. URL: https://dslpitt.org/papers/02/p311-madani.pdf.

[Mar75] Donald A. Martin. “Borel determinacy”. In: *Annals of Mathematics* 102(2) (1975), pp. 363–371.

[McC93] S. Thomas McCormick. “Approximate Binary Search Algorithms for Mean Cuts and Cycles”. In: *Operations Research Letters* 14.3 (1993), pp. 129–132. DOI: 10.1016/0167-6377(93)90022-9.

[McN93] Robert McNaughton. “Infinite games played on finite graphs”. In: *Annals of Pure and Applied Logic* 65.2 (1993), pp. 149–184. DOI: 10.1016/0168-0072(93)90036-D.

[Mil13] Peter Bro Miltersen. “Recent Results on Howard’s Algorithm”. In: *Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS).* 2013, pp. 53–56. DOI: 10.1007/978-3-642-36046-6_6.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* New York: Springer-Verlag, 1992. DOI: 10.1007/978-1-4612-0931-7.

[OA92] James B. Orlin and Ravindra K. Ahuja. “New scaling algorithms for the assignment and minimum mean cycle problems”. In: *Mathematical Programming* 54.1-3 (1992), pp. 41–56. DOI: 10.1007/BF01586040.

[Pat10] Mihai Patrascu. “Towards polynomial lower bounds for dynamic problems”. In: *Symposium on Theory of Computing (STOC).* 2010, pp. 603–610. DOI: 10.1145/1806689.1806772.

[PPS06] Nir Piterman, Amir Pnueli, and Yaniv Sa’ar. “Synthesis of Reactive(1) Designs”. In: *International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI).* 2006, pp. 364–380. DOI: 10.1007/11609773_24.

[PR89] Amir Pnueli and Roni Rosner. “On the synthesis of a reactive module”. In: *Symposium on Principles of Programming Languages (POPL).* 1989, pp. 179–190. DOI: 10.1145/75277.75293.

[PSL00] Anna Pogosyants, Roberto Segala, and Nancy A. Lynch. “Verification of the Randomized Consensus Algorithm of Aspnes and Herlihy: a Case Study”. In: *Distributed Computing* 13.3 (2000), pp. 155–186. DOI: 10.1007/PL00008917.

[PW10] Mihai Patrascu and Ryan Williams. “On the Possibility of Faster SAT Algorithms”. In: *Symposium on Discrete Algorithms (SODA).* 2010, pp. 1065–1075. DOI: 10.1137/1.9781611973075.86.

[QS82] Jean-Pierre Queille and Joseph Sifakis. “Specification and verification of concurrent systems in CESAR”. In: *International Symposium on Programming.* 1982, pp. 337–351. DOI: 10.1007/3-540-11494-7_22.

[Ras16] Jean-François Raskin. “A Tutorial on Mean-payoff and Energy Games”. In: *Dependable Software Systems Engineering*. 2016, pp. 179–201. DOI: 10.3233/978-1-61499-627-9-179.

[Rot+10] Aaron Roth, Maria-Florina Balcan, Adam Kalai, and Yishay Mansour. “On the Equilibria of Alternating Move Games”. In: *Symposium on Discrete Algorithms (SODA)*. 2010, pp. 805–816. DOI: 10.1137/1.9781611973075.66.

[RW87] P.J. Ramadge and W. Murray Wonham. “Supervisory control of a class of discrete-event processes”. In: *SIAM Journal of Control and Optimization* 25.1 (1987), pp. 206–230. DOI: 10.1137/0325013.

[RZ11] Liam Roditty and Uri Zwick. “On Dynamic Shortest Paths Problems”. In: *Algorithmica* 61.2 (2011), pp. 389–401. DOI: 10.1007/s00453-010-9401-5. Announced at ESA’04.

[Saf88] Shmuel Safra. “On the complexity of ω-automata”. In: *Symposium on Foundations of Computer Science (FOCS)*. 1988, pp. 319–327. DOI: 10.1109/SFCS.1988.21948.

[Saf89] Shmuel Safra. “Complexity of automata on infinite objects”. PhD thesis. Weizmann Institute of Science, 1989.

[Saf92] Shmuel Safra. “Exponential Determinization for omega-Automata with Strong-Fairness Acceptance Condition (Extended Abstract)”. In: *Symposium on Theory of Computing (STOC)*. 1992, pp. 275–282. DOI: 10.1145/129712.129739.

[San05] Piotr Sankowski. “Shortest Paths in Matrix Multiplication Time”. In: *European Symposium on Algorithms (ESA)*. 2005, pp. 770–778. DOI: 10.1007/11561071_68.

[Sch07] Sven Schewe. “Solving Parity Games in Big Steps”. In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 2007, pp. 449–460. DOI: 10.1007/978-3-540-77050-3_37.

[Sch08] Sven Schewe. “Synthesis of Distributed Systems”. PhD thesis. Universität des Saarlandes, 2008.

[Sch17] Sven Schewe. “Solving Parity Games in Big Steps”. In: *Journal of Computer and Systems Science* (2017). To appear.

[Sei96] Helmut Seidl. “Fast and Simple Nested Fixpoints”. In: *Information Processing Letters* 59.6 (1996), pp. 303–308. DOI: 10.1016/0020-0190(96)00130-5.

[Sip06] Michael Sipser. *Introduction to the Theory of Computation*. Vol. 2. Boston: Thomson Course Technology, 2006.

[Som15] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 3.0.0*. 2015. URL: http://vlsi.colorado.edu/~fabio/CUDD/.

[Sto03] Mariëlle Stoelinga. “Fun with FireWire: A Comparative Study of Formal Verification Methods Applied to the IEEE 1394 Root Contention Protocol”. In: *Formal Aspects of Computing* 14.3 (2003), pp. 328–337. DOI: 10.1007/s001650300009.

[Tar72] Robert E. Tarjan. “Depth First Search and Linear Graph Algorithms”. In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010.

[Tho95] Wolfgang Thomas. “On the Synthesis of Strategies in Infinite Games”. In: *Symposium on Theoretical Aspects of Computer Science (STACS).* 1995, pp. 1–13. DOI: 10.1007/3-540-59042-0_57.

[Tho97] Wolfgang Thomas. “Languages, Automata, and Logic”. In: *Handbook of Formal Languages: Volume 3 Beyond Words.* Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer, 1997, pp. 389–455. DOI: 10.1007/978-3-642-59126-6_7.

[Tho98] Mikkel Thorup. “All Structured Programs Have Small Tree Width and Good Register Allocation”. In: *Information and Computation* 142.2 (1998), pp. 159–181. DOI: 10.1006/inco.1997.2697.

[Var85] Moshe Y. Vardi. “Automatic Verification of Probabilistic Concurrent Finite-State Programs”. In: *Symposium on Foundations of Computer Science (FOCS).* 1985, pp. 327–338. DOI: 10.1109/SFCS.1985.12.

[Var96] Moshe Y. Vardi. “An Automata-Theoretic Approach to Linear Temporal Logic”. In: *Banff 1995.* 1996, pp. 238–266. DOI: 10.1007/3-540-60915-6_6.

[Vas12] Virginia Vassilevska Williams. “Multiplying Matrices Faster Than Coppersmith-Winograd”. In: *Symposium on Theory of Computing (STOC).* 2012, pp. 887–898. DOI: 10.1145/2213977.2214056.

[Vas15] Virginia Vassilevska Williams. “Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk)”. In: *International Symposium on Parameterized and Exact Computation (IPEC).* 2015, pp. 17–29. DOI: 10.4230/LIPIcs.IPEC.2015.17.

[VJ00] Jens Vöge and Marcin Jurdziński. “A Discrete Strategy Improvement Algorithm for Solving Parity Games”. In: *International Conference on Computer Aided Verification (CAV).* 2000, pp. 202–215. DOI: 10.1007/10722167_18.

[VW10] Virginia Vassilevska Williams and Ryan Williams. “Subcubic Equivalences between Path, Matrix and Triangle Problems”. In: *Symposium on Foundations of Computer Science (FOCS).* 2010, pp. 645–654. DOI: 10.1109/FOCS.2010.67.

[VW13] Virginia Vassilevska Williams and Ryan Williams. “Finding, Minimizing, and Counting Weighted Subgraphs”. In: *SIAM Journal on Computing* 42.3 (2013), pp. 831–854. DOI: 10.1137/09076619X. Announced at STOC’09.

[VW86] Moshe Y. Vardi and Pierre Wolper. “An Automata-Theoretic Approach to Automatic Program Verification”. In: *Symposium on Logic in Computer Science (LICS).* 1986, pp. 322–331.

[Wal00] Igor Walukiewicz. “Completeness of Kozen’s Axiomatisation of the Propositional μ-Calculus”. In: *Information and Computation* 157.1-2 (2000), pp. 142–182. DOI: 10.1006/inco.1999.2836.

[Wil05] Ryan Williams. “A new algorithm for optimal 2-constraint satisfaction and its implications”. In: *Theoretical Computer Science* 348.2-3 (2005), pp. 357–365. DOI: 10.1016/j.tcs.2005.09.023. Announced at ICALP’04.

[Wil14a] Ryan Williams. “Faster all-pairs shortest paths via circuit complexity”. In: *Symposium on Theory of Computing (STOC).* 2014, pp. 664–673. DOI: 10.1145/2591796.2591811.

[Wil14b] Ryan Williams. “Faster decision of first-order graph properties”. In: *Joint Meeting of Conference on Computer Science Logic and Symposium on Logic in Computer Science (CSL-LICS)*. 2014, 80:1–80:6. DOI: 10.1145/2603088.2603121.

[Woe04] Gerhard J. Woeginger. “Space and Time Complexity of Exact Algorithms: Some Open Problems (Invited Talk)”. In: *International Workshop on Parameterized and Exact Computation (IWPEC)*. 2004, pp. 281–290. DOI: 10.1007/978-3-540-28639-4_25.

[Wol00] Pierre Wolper. “Constructing Automata from Temporal Logic Formulas: A Tutorial”. In: *Lectures on Formal Methods and Performance Analysis*. 2000, pp. 261–277. DOI: 10.1007/3-540-44667-2_7.

[WT05] Chi-Him Wong and Yiu-Cheong Tam. “Negative Cycle Detection Problem”. In: *European Symposium on Algorithms (ESA)*. 2005, pp. 652–663. DOI: 10.1007/11561071_58.

[YTO91] Neal E. Young, Robert E. Tarjan, and James B. Orlin. “Faster Parametric Shortest Path and Minimum-Balance algorithms”. In: *Networks* 21.2 (1991), pp. 205–221. DOI: 10.1002/net.3230210206.

[Yu15] Huacheng Yu. “An Improved Combinatorial Algorithm for Boolean Matrix Multiplication”. In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 2015, pp. 1094–1105. DOI: 10.1007/978-3-662-47672-7_89.

[Yuv76] Gideon Yuval. “An algorithm for finding all shortest paths using $N^{2.81}$ infinite-precision multiplications”. In: *Information Processing Letters* 4.6 (1976), pp. 155–156. DOI: 10.1016/0020-0190(76)90085-5.

[Zem87] Eitan Zemel. “A Linear Time Randomizing Algorithm for Searching Ranked Functions”. In: *Algorithmica* 2.1-4 (1987), pp. 81–90. DOI: 10.1007/BF01840350.

[Zie98] Wieslaw Zielonka. “Infinite games on finitely coloured graphs with applications to automata on infinite trees”. In: *Theoretical Computer Science* 200.1–2 (1998), pp. 135–183. DOI: 10.1016/S0304-3975(98)00009-7.

[ZP96] Uri Zwick and Mike Paterson. “The complexity of mean payoff games on graphs”. In: *Theoretical Computer Science* 158.1-2 (1996), pp. 343–359. DOI: 10.1016/0304-3975(95)00188-3. Announced at COCOON ’95.

[Zwi02] Uri Zwick. “All Pairs Shortest Paths using Bridging Sets and Rectangular Matrix Multiplication”. In: *Journal of the ACM* 49.3 (2002), pp. 289–317. DOI: 10.1145/567112.567114. Announced at FOCS ’98.