Multidimensional and Pushdown Mean-Payoff Games with Applications to Quantitative Verification and Synthesis of Programs: Complexity, Decidability and Algorithms

Thesis submitted for the degree of Doctor of Philosophy
by
Yaron Velner

This work was carried out under the supervision of
Professor Alexander Rabinovich

Submitted to the Senate of Tel Aviv University
July 2015
Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Alexander Rabinovich, for giving me freedom to choose my trail, for raising the bar high, and for teaching me how to write accurate proofs. Alex introduced me to the field of formal verification and graph games. His guidance in the last six years, during my master and doctoral studies, and his unwillingness to be impressed by easy results encouraged me to tackle the non-trivial problems and to improve the quality of my work.

I am also thankful to Prof. Krishnendu Chatterjee. The collaborative works with Krish accounts to almost half of this manuscript. Working with him was tremendously enriching and encouraging. His thorough approach and attention for details (e.g., when revising a journal version three years after the first submission) are really inspiring. I am also grateful for his hospitality and warm welcome during my numerous visits to the IST of Austria.

I want to thank EZchip Semiconductor for allowing me to pursue my master and doctoral degrees while working there. For their flexibility and understanding towards my absences for conferences, collaborations abroad and others.

Finally, I would like to thank my family:

My parents, Shimshon and Liora, for their constant support and encouragement during the years of research, and not only to thesis related matters.

My wife Zelda for supporting me and helping me. Zelda has always been there for me whenever I needed her. When I had to make important decisions, when I found bugs in my proofs in the middle of the night, or when I complained for no reason. And for helping me to produce the best result of my studies: our daughter Sivan.

My grandfather, Yosef Ulizky, always urged his grandsons to pursue higher education. Yosef passed away, at the age of 95, two months before this manuscript was officially approved. This thesis is dedicated to his memory.
Abstract

Games on graphs play a central role in the automated verification and synthesis of programs. In the recent years there is an extensive line of research that aims to extend traditional boolean verification with quantitative properties. Typically, quantitative properties are modelled by a weight function over the transitions of the program. The most well-studied quantitative metric is the mean-payoff metric which assigns to a run of the program the mean-payoff (long-run average) of the weights that occur in the run.

The automated synthesis of a program is modeled by infinite-duration two-player games over graphs, and a winning strategy for the protagonist corresponds to a correct implementation of the program. The automated verification of a program is modeled by a one-player game, where only one player makes the choices, and the existence of a winning strategy (i.e., an infinite path in the graph) corresponds to undesirable run of the program (and therefore, the implementation of the program is incorrect).

Mean-payoff games are played over weighted graphs, and the vertices of the graph are partitioned among the protagonist and the antagonist. The player who owns the current vertex decides on the next move, and the aim of the protagonist is to get a non-negative mean-payoff value for the play. The study of these games was initially motivated by applications to economics, but recently they are extensively used for the study of quantitative verification and synthesis of programs.

The purpose of this dissertation is to extend the classical research of mean-payoff games in two orthogonal directions:

The first direction is to consider games over finite-graphs with multidimensional weight function. A multidimensional weight function can model multiple (and possibly conflicting) properties of a program, for example energy consumption and response time. In multidimensional mean-payoff games the objective of the protagonist is to satisfy a boolean condition over the mean-payoff values of the different dimensions. For example, achieve mean-payoff value at least 7 in the first dimension, while maintaining either mean-payoff of at least -2 in the second dimension or mean-payoff of at most 4 in the third dimension.

The second direction is to consider games over infinite-state pushdown graphs, namely, games over the state space of pushdown automata that can model the control flow of sequential programs with recursion.

The main contributions of this thesis are as follows.

(i) Multidimensional mean-payoff games on finite graphs. We obtain a complete characterization of the complexity and decidability of multidimensional mean-payoff
games. Specifically, we obtain tight complexity and decidability results for: (a) Two-player games with conjunctive condition (e.g., assure non-negative mean-payoff in every dimension) and bounded number of dimensions. (b) One-player games with arbitrary boolean condition. (c) Two-player games with arbitrary condition, where the protagonist is restricted to finite-memory strategies. (d) Two-player games with arbitrary condition and arbitrary strategies.

(ii) \textit{Mean-payoff pushdown games.} In pushdown games two types of strategies are relevant: global strategies, that depend on the entire global history; and modular strategies, that have only local memory and thus do not depend on the context of invocation.

a) \textit{Global strategies.} We show that two-player games are undecidable. Hence, we focus on one-player games and give a polynomial algorithm. In addition, we show that one-player multidimensional pushdown mean-payoff games with conjunctive objective also have a polynomial solution.

b) \textit{Modular strategies.} We show that two-player games are decidable and give tight complexity bounds for solving mean-payoff pushdown games (when the protagonist is restricted to modular strategies).

(iii) \textit{Applications.} We show two applications for our results on multidimensional and infinite-state games. One application resides in the field of theoretical game theory where games are equipped with payoff matrices (rather than a payoff function that is represented by a graph), and one application is in the field of static program analysis.

a) We apply the techniques for solving one-player multidimensional mean-payoff games on graphs and settle the complexity of finding an exact and optimal equilibrium in a multiplayer infinite-duration mean-payoff game (where the payoff of each move is represented by a payoff matrix).

b) We apply and implement our one-player mean-payoff pushdown game algorithm for static interprocedural analysis that reason about quantitative properties of programs and present several concrete applications and case studies.
# Contents

1 Introduction ................................................................. 1  
   1.1 Background ......................................................... 1  
   1.2 Main Contribution and Related Works ............................... 3  
      1.2.1 Multidimensional mean-payoff games on finite-graphs ....... 4  
      1.2.2 Mean-payoff pushdown games ................................ 7  
      1.2.3 Applications .................................................. 8  
   1.3 Outline of Thesis and Related Publications ....................... 10  

I Multidimensional Mean-Payoff Games on Finite Graphs ................. 12  

2 Definitions and Outline .................................................. 13  
   2.1 Definitions ....................................................... 13  
   2.2 Outline ............................................................ 15  

3 Hyperplane Separation Technique for Conjunctive Multidimensional  
Mean-Payoff Games ................................................................ 16  
   3.1 Definitions ............................................................ 17  
   3.2 Hyperplane separation algorithm .................................... 17  
   3.3 Hardness for fixed parameter tractability ........................... 25  

4 The Complexity of Model-Checking Problems with Mean-Payoff Ex-  
pression Specifications ....................................................... 27  
   4.1 Mean-Payoff Automaton Expression ................................. 28  
   4.2 PSPACE Algorithm for Computing the Maximum Value of max-free  
      Expressions .................................................................. 30  
      4.2.1 The emptiness problem for intersection of LimSupAvg automata  
      .................................................................................. 30  
      4.2.2 The emptiness problem for intersection of LimInfAvg automata .... 31  
      4.2.3 The emptiness problem for intersection of LimSupAvg  
      and LimInfAvg automata ............................................... 34  
      4.2.4 The emptiness problem for max-free expressions ............... 36  
      4.2.5 PSPACE algorithm for the emptiness problem of max-free ex-  
      pressions ..................................................................... 37  
   4.3 The Complexity of Mean-Payoff Expression Problems .......... 42
5 Finite-Memory Strategy Synthesis for Multidimensional Mean-Payoff Objectives

5.1 Games with Quantitative Objectives

5.1.1 Quantitative games on graphs

5.1.2 One-player game solution

5.1.3 Informal overview of the solution for two-player games

5.2 CONVEX Cycles Problem

5.3 Generic Solution for Games with Quantitative Objectives

5.4 Games with Mean-Payoff Expression Objectives

5.4.1 Mean-payoff expression objectives

5.4.2 Synthesis of a finite-memory controller for mean-payoff expression objectives

5.5 Proof of Lemma 26

5.5.1 Alternative formulations of $H_{10}(Q)$

5.5.2 Auxiliary lemma

5.5.3 The reduction

5.6 Discussion

6 Multidimensional Mean-Payoff Games are Undecidable

6.1 Notations and Definitions

6.2 Reduction from the Halting Problem and Informal Proof of Correctness

6.3 Detailed Proof

6.3.1 If $M$ halts, then player 2 is the winner

6.3.2 If $M$ does not halt, then player 1 is the winner

6.3.3 Extending the reduction to two-counter machine

6.4 Discussion

II Mean-Payoff Pushdown Games

7 Mean-Payoff Pushdown Games

7.1 Mean-Payoff Pushdown Graphs

7.1.1 Objectives $\liminf_{\infty} > 0$ and $\limsup_{\infty} > 0$

7.1.2 Objectives $\liminf_{\infty} \geq 0$ and $\limsup_{\infty} \geq 0$

7.1.3 Mean-payoff objectives with stack boundedness

7.2 Pushdown Graphs with (Conjunctive) Multidimensional Mean-payoff Objectives

7.2.1 Technical detailed proof of Proposition 4

7.3 Mean-Payoff Pushdown Games
8 Pushdown Mean-Payoff Games with Modular Strategies 134
8.1 Recursive Games and Modular Strategies 134
8.1.1 Decidability of the modular winning strategy problem 137
8.1.2 Modular winning strategy problem in NP 147
8.1.3 NP-hardness of the modular winning strategy problem 155
8.2 Undecidability for (Conjunctive) Multidimensional Mean-Payoff Objectives 157

III Applications 161

9 The Complexity of Infinitely Repeated Alternating Move Games 162
9.1 Definitions 163
9.1.1 Alternating move repeated games 163
9.1.2 Mean-payoff games on graphs 164
9.2 Two-Player Zero-Sum (Alternating Move) Games are Inter-Reducible with Mean-Payoff Games 166
9.3 Complexity of Computing Optimal Equilibrium 169
9.3.1 Basic properties of equilibria 170
9.3.2 Optimal equilibrium can be obtained by pure strategies 171
9.3.3 The complexity of computing the social welfare of the optimal (exact) equilibrium 173
9.3.4 An FPTAS to compute an $\epsilon$-equilibrium that is $\delta$-optimal 174

10 Quantitative Interprocedural Analysis 178
10.1 Definitions 182
10.2 Applications: Theoretical Modeling 185
10.2.1 Container analysis 185
10.2.2 Static profiling 188
10.2.3 Estimating worst-case execution time 188
10.2.4 Evaluating the speedup in a parallel computation 189
10.2.5 Average energy consumption 189
10.3 Algorithm for Quantitative Analysis of QICFGs 189
10.3.1 Improved algorithm for mean-payoff analysis 190
10.3.2 Efficient algorithm for mean-payoff analysis 198
10.3.3 Reduction: Ratio analysis to mean-payoff analysis 199
10.4 Experimental Results: Two Case Studies 201
10.4.1 Optimization for case studies 201
10.4.2 Container analysis 203
10.4.3 Static profiling: frequency of function calls 205
10.5 Related Work 208
10.6 Conclusion 209
Chapter 1

Introduction

In this chapter we introduce the background and the contributions of this dissertation. In the next section we give a brief survey on automated verification and games on graphs. In Section 1.2 we give an overview on the main contributions of our work and compare our results with the relevant related works. In Section 1.3 we present the structure of this thesis.

Bibliographic note. Most of the research presented in this thesis was published in conference proceedings. In Section 1.2 we mention where each of our contribution was published, and in Section 1.3 we give a full list of our publications.

1.1 Background

In this section we give the necessary background on the automated verification and synthesis tasks, and the connection between these tasks and infinite-duration games on graphs. We survey the recent emerging interest in quantitative verification and synthesis, and present mean-payoff games.

Automated verification and synthesis, and games on graphs. The task of automatically verify the correctness of a given program with respect to a given formal specification lies in the heart of formal methods and automata theory. Tools for automated verifications are extensively used by software and hardware vendors, e.g., [3, 60, 62]. A more ambitious task is to automatically synthesize (construct) a system that is correct by design with respect to a formal specification.

In formal methods, the verification task (a.k.a model checking problem) is solved by modeling the system and the specification with an automaton and an accepting condition, and a non-accepting path in the underlining graph of the automaton implies that there is a potential run of the program that do not meet the specification, and thus the implementation of the program is not correct. The synthesis task is modelled by an infinite-duration two-player game on graphs. In such games, the vertex set of the graph is partitioned into player-1 (system) vertices and player-2 (environment) vertices. The game starts at an initial vertex, and if the current vertex is a player-1 vertex, then player 1 chooses an outgoing edge, and if the current vertex is a player-2 vertex, then player 2 does likewise. This process is repeated forever, and gives rise to an outcome of the game, called a play, that consists of the infinite sequence of vertices that are visited. Player 1
wins the game if the formed sequence (which corresponds to a run of the program) meets the specification. A strategy of a player is a recipe for deciding the next move in a play, and a player-1 strategy that assures his win against any player-2 strategy is called a winning strategy. As player 2 strategies correspond to all possible inputs of the program, a player 1 winning strategy corresponds to a correct implementation of the program. Hence, the automated synthesis boils down to automatically compute player-1 winning strategy in a two-player game. In the framework of games on graphs, the model checking problem amounts to a one-player game in which one player owns all the vertices of the graph, and thus, only one player decides on the moves in the game. Indeed, in one-player games, a strategy is an infinite path in the graph. Hence, a witness violating path is a winning strategy for player 2.

The classical works on model checking and automated synthesis consider boolean specifications such as liveness and safety properties. Two notable landmarks are the work of Pnueli [79] that introduced the use of temporal logic specifications for automated verification, and the Büchi-Landweber Theorem [27] that proves the computability of the synthesis task for \( \omega \)-regular specifications.

Quantitative verification and synthesis. In recent years, there is an emerging line of research that aims to formalize and reason about the quality of software and hardware systems, e.g., [7, 21, 25, 33, 34, 45]. As opposed to traditional verification, where one handles the question of whether a system satisfies, or not, a given specification, reasoning about quality addresses the question of how well the system satisfies the specification. This line of research introduces a paradigm shift from boolean verification to quantitative verification [56]. A quantitative specification assigns a numerical value to every run of the program (rather than a boolean value). The quantitative verification problem is to measure how well a program satisfies a given quantitative specification, e.g., to compute the minimal value of all possible runs of the program. The quantitative synthesis problem is to automatically construct an optimal program with respect to a given quantitative specification. Quantitative properties are typically modeled by weighted automata [34] and the most well studied quantitative metric is the mean-payoff metric which assigns to a run (over a weighted automaton) its limit-average (long-run average) sum of weights.

Mean-payoff games on graphs. Mean-payoff games are played on weighted finite-graphs. A weight (representing a reward) is associated with every transition and the goal of one of the players is to maximize the long-run average of the weights; and the goal of the opponent is to minimize. Mean-payoff games and the special case of graphs (with only one player) with mean-payoff objectives have been extensively studied over the last three decades; e.g. [47, 54, 61, 110]. Graphs with mean-payoff objectives can be solved in polynomial time [61], whereas mean-payoff games can be decided in \( \text{NP} \cap \text{coNP} \) [47, 110]. The mean-payoff games problem is an intriguing problem and one of the rare combinatorial problems that is known to be in \( \text{NP} \cap \text{coNP} \), but no polynomial time algorithm is known. However, pseudo-polynomial time algorithms exist for mean-
payoff games [26, 110], and if the weights are bounded by a constant, then the algorithm is polynomial.

1.2 Main Contribution and Related Works

In this section we describe our main contributions and survey the related works. We first state the purpose of this dissertation and give an overview on the main contributions. Then we give a detailed description of our results and compare them to the known related works.

The purpose of this thesis is to extend the classical study on classical mean-payoff games, and thus, to extend the known tools and building blocks that are used for automated quantitative verification and synthesis of systems.

Motivated by applications in formal analysis of reactive systems, we extend the study of mean-payoff games in two orthogonal directions: (1) multidimensional mean-payoff games on finite game graphs; and (2) pushdown mean-payoff games.

In applications of verification and synthesis, the quantitative objectives that typically arise are multidimensional quantitative objectives (e.g., conjunction of several objectives), e.g., to express properties like the average response time between a grant and a request is below a given threshold 1, and the average number of unnecessary grants is below a threshold 2. Thus mean-payoff objectives can express properties related to resource requirements, performance, and robustness; and multiple objectives can express the different, potentially dependent or conflicting objectives. Moreover, recently many quantitative logics and automata theoretic formalisms have been proposed with mean-payoff objectives in their heart to express properties such as reliability requirements, and resource bounds of reactive systems [20, 21, 34, 44], and several quantitative synthesis questions (such as synthesis from incompatible specifications [29]) translate directly to multidimensional mean-payoff games.

Pushdown games, a.k.a games on recursive state machines, can model reactive systems with recursion (i.e., model the control flow of sequential programs with recursion). In the boolean framework, pushdown games have been studied widely with applications in verification, synthesis, and program analysis in [8, 9, 81, 101, 102]. Thus pushdown games and graphs with mean-payoff objectives, and finite-state game graphs with multidimensional mean-payoff objectives are fundamental theoretical questions in model checking of quantitative logics and quantitative analysis of reactive systems (along with recursion features).

In addition, we show two applications for our study of multidimensional and pushdown mean-payoff games. The first application resides in the field of theoretical game theory. We rely on the solution for one-player multidimensional mean-payoff games to settle the complexity of finding an exact and optimal equilibrium in a multi-player alternating move mean-payoff games with payoff matrices (i.e., the payoff function is given as a matrix, rather than a graph). The second application is in the area of static program analysis. We apply and implement our one-player mean-payoff pushdown game algorithm for static interprocedural analysis that reason about quantitative
properties of programs and present several case studies and concrete applications.

We present our contributions with more details in the next three subsections. In the first subsection we describe our results for multidimensional finite-graph mean-payoff games, in the second subsection we describe the results on mean-payoff pushdown games, and in the last subsection we describe the applications.

1.2.1 Multidimensional mean-payoff games on finite-graphs

Multi-objective mean-payoff games are modeled by multi-weighted graphs (that is, a weight vector is assigned to every edge in the graph). The outcome of a play on a \( k \)-dimensional graph is a vector of \( k \) mean-payoffs, where the mean-payoff of dimension \( i \) is determined by the projection of the sequence of weights to the \( i \)-th dimension. The goal of the protagonist player is either to satisfy a boolean condition or to maximize a quantitative expression over the vector of payoffs.

A boolean condition on the vector of payoffs \((p_1, \ldots, p_k)\) is given by a vector of thresholds \((t_1, \ldots, t_k)\) and a boolean formula

\[
\varphi(p_1 > t_1, p_1 \geq t_1, \ldots, p_k > t_k, p_k \geq t_k)
\]

for example: \((p_1 \geq 2) \land (p_2 \geq 7) \lor (p_1 < 9)\) (note that \((p_1 < 9) \equiv \neg(p_1 \geq 9))\).

An interesting special case is the conjunctive condition that requires all payoffs to be non-negative (formally the condition is \( \bigwedge_{i=1}^k p_i \geq 0 \)). We note that any condition of the form \( \bigwedge_{i=1}^k p_i \geq t_i \) over a graph \( G \) and a weight function \( w \) is equivalent to the condition \( \bigwedge_{i=1}^k p_i \geq 0 \) over the same graph with a weight function \( w' = w - (t_1, \ldots, t_k) \). Hence, the conjunctive condition is a canonical form for all the boolean conditions that contain only the conjunction operator.

A quantitative expression is a function that assigns a real value to a payoff vector. The quantitative analogue of the basic boolean operations: disjunction, conjunction and complement, are the max, min and numerical complement (multiplication by \(-1\)) operators; and another important operator is the sum operator. With these operators we recursively define the class of mean-payoff expressions as follows:

- A projection of the mean-payoff vector to a single dimension is an expression (e.g., \( p_7 \)).
- If \( E_1 \) and \( E_2 \) are expressions, then \( \text{op}(E_1, E_2) \) is also an expression (for \( \text{op} \in \{\max, \min, +, -\} \)).

E.g., a possible expression is

\[
\max(p_1, p_2 + \min(p_3 - p_4))
\]

A special interesting case is to consider the objective \( E = \min(p_1, \ldots, p_k) \), and it is straightforward that the protagonist can win for the boolean objective that requires
all payoffs to be non-negative if and only if the maximum value that he can assure for 
\( E \) is non-negative.

1.2.1.1 Related work.

The model checking problem (one-player game) for multidimensional objectives was 
first considered by Alur et al. [7] who considered arbitrary boolean conditions over 
mean-payoff vectors. Boker et al. [21] and Tomita et al. [93] considered a more general 
objective with combination of boolean conditions over mean-payoff vectors and standard \( \omega \)-regular objectives. Chatterjee et al. [31] introduced the quantitative objective 
of mean-payoff expressions and solved the corresponding model-checking problem.

Two-player games over the conjunctive condition were studied in the author master 
thesis and in [20, 37, 98] and the problem of determining whether the protagonist has 
a winning strategy was proved to be coNP-complete. Two-player games over general 
boolean formulas (or over general mean-payoff expressions) were never investigated.

1.2.1.2 Our contribution.

1. Hyperplane technique for multidimensional mean-payoff games with conjunctive 
condition. We use the separating hyperplane technique from computational geom-
etry to solve two-player mean-payoff games over conjunctive condition (e.g., assure 
that the mean-payoff in all dimensions is non-negative). Our technique reduces 
the multidimensional problem to searching for a separating hyperplane such that 
all realizable mean-payoff vectors lie on one side of the hyperplane. This technique 
allows us to search for a vector, which is normal to the separating hyperplane, and 
reduce the multidimensional problem to a one-dimensional problem by multiply-
ing the multidimensional weight function by the vector. The main contribution 
of this technique is an algorithm that runs in 
\( O(n^2 \cdot m \cdot k \cdot W \cdot (k \cdot n \cdot W)^{k^2+2^k+1}) \) 
time (where \( n \) denotes the number of vertices in the graph, \( m \) denotes the num-
ber of edges, \( W \) denotes the maximal weight in the graph and \( k \) is the number 
of dimensions). Thus, the algorithm is polynomial when the weights and number 
of dimensions are fixed. Prior to this work, a (pseudo) polynomial algorithm was 
not known even for \( k = 2 \). In addition, our technique gives a simple and elegant 
description for the (infinite-memory) winning strategy of the protagonist.

Related publication: Krishnendu Chatterjee, Yaron Velner: Hyperplane Separation Technique for Multidimensional Mean-Payoff Games. CONCUR 2013.

2. Model checking complexity for mean-payoff expressions. The decidability of the 
model checking problem over mean-payoff expressions was established by Chatter-
jee et al. in [31]. In this work we give a better algorithm for the problem. Our ap-
proach offer the following advantages: First, the algorithm yields PSPACE com-
plexity upper bound, which matches corresponding PSPACE lower bounds, while 
the previously known algorithm gives 4EXPTIME upper bound. Second, our 
proofs reside only in the frameworks of graph theory and basic linear-programing,
which are common practices among the automata-theoretic community, whereas a substantial part of the proofs in [31] resides in the framework of computational geometry.


3. Finite-memory strategy synthesis for mean-payoff expressions. We consider multidimensional mean-payoff games when the protagonist is restricted to play with finite-memory strategies. As the winning strategy of the protagonist correspond to a correct implementation of a system, the restriction to finite-memory strategies is natural when one wants to synthesis simple implementations (e.g., for hardware implementation).

In this framework, there are two relevant synthesis problems to consider: the quantitative analysis problem is to compute the maximal (or supremum) value that the protagonist can assure, and the boolean analysis problem asks whether the protagonist can assure that the value of the objective is at most $\nu$ (for a given threshold $\nu$).

In this thesis, we consider for the first time the synthesis problem for a robust class of quantitative objectives, namely, the class of mean-payoff expressions. We prove computability for the quantitative synthesis problem, and we show that the boolean analysis problem is inter-reducible with Hilbert’s tenth problem over rationals ($H10(\mathbb{Q})$), which is a fundamental long-standing open question in computer science and mathematics. We show that the problem is inter-reducible with $H10(\mathbb{Q})$ even when both players are restricted to finite-memory strategies, and we show that there is a fragment of mean-payoff expressions that is $H10(\mathbb{Q})$-hard when one or both players are restricted to finite-memory strategies, but decidable when both players may use infinite-memory strategies.

Our main technical contribution is the introduction of a general scheme that lifts a one-player game solution (equivalently, a model checking algorithm) to a solution for a two-player game (when player 1 is restricted to finite-memory strategies). The scheme works for a large class of quantitative objectives that have certain properties.


4. Undecidability of games over mean-payoff expressions with infinite-memory strategies. We consider games over mean-payoff expressions and allow both players to play with arbitrary (infinite-memory) strategies. We show that in this case, the problem of determining who is the winner is undecidable.

Related publication: Yaron Velner: Robust Multidimensional Mean-Payoff Games are Undecidable. FOSSACS 2015.
CHAPTER 1. INTRODUCTION

1.2.2 Mean-payoff pushdown games

The classical study of two-player finite-state games with boolean objectives has been extended in two orthogonal directions in the literature: (1) two-player infinite-state games with qualitative objectives; and (2) two-player finite-state games with quantitative objectives. One of the most well-studied models of infinite-state games with qualitative objectives is pushdown games (or games on recursive state machines) that can model reactive systems with recursion (or model the control flow of sequential programs with recursion). Pushdown games with reachability and parity objectives have been studied in [8, 9, 101, 102] (also see [23, 24, 49, 50] for sample research in stochastic pushdown games). The most well-studied quantitative objective is the mean-payoff objective. Thus pushdown games with mean-payoff objectives would be a central theoretical question for model checking of quantitative logics (specifying reliability and resource bounds) on reactive systems with recursion feature.

In this work we study for the first time pushdown games with mean-payoff objectives. In pushdown games two types of strategies are relevant and studied in the literature. The first are the global strategies, where a global strategy can choose the successor state depending on the entire global history of the play (where history is the finite sequence of configurations of the current prefix of a play). The second are the modular strategies, and modular strategies are understood more intuitively in the model of games on recursive state machines. A recursive state machine (RSM) consists of a set of component machines (or modules). Each module has a set of nodes (atomic states) and boxes (each of which is mapped to a module), a well-defined interface consisting of entry and exit nodes, and edges connecting nodes/boxes. An edge entering a box models the invocation of the module associated with the box and an edge leaving the box represents return from the module. In the game version the nodes are partitioned into player-1 nodes and player-2 nodes. Due to recursion the underlying global state-space is infinite and isomorphic to pushdown games. A modular strategy is a strategy that has only local memory, and thus, the strategy does not depend on the context of invocation of the module, but only on the history within the current invocation of the module. In other words, modular strategies are appealing because they are stackless strategies, decomposable into one for each module. In this work we study pushdown games with mean-payoff objectives for both global and modular strategies.

1.2.2.1 Previous work.

Pushdown games with qualitative objectives were studied in [101, 102]. It was shown in [102] that solving pushdown games (i.e., determining the winner in pushdown games) with reachability objectives under global strategies is \(\text{EXPTIME-hard,}\) and pushdown games with parity objectives under global strategies can be solved in \(\text{EXPTIME.}\) Thus it follows that pushdown games with reachability and parity objectives under global strategies are \(\text{EXPTIME-complete.}\) The notion of modular strategies in games on recursive state machines was introduced in [8, 9]. It was shown that the modular
strategies problem is NP-complete in pushdown games with reachability and parity objectives in general [8, 9]. The results of [9] also presents more refined complexity results in terms of the number of exit nodes, showing that if every module has single exit, then the problem is polynomial for reachability objectives [9] and in $\text{NP} \cap \text{coNP}$ for parity objectives [8].

1.2.2.2 Our contributions.

In this work, together with Krishnendu Chatterjee, we present a complete characterization of the computational and strategy complexity of pushdown games and pushdown systems (one-player pushdown games or pushdown automata) with mean-payoff objectives. Solving a pushdown system (resp. pushdown game) with respect to a mean-payoff objective is to decide whether there exists a path that (resp. a winning strategy to ensure that every path possible given the strategy) satisfies the mean-payoff objective. Our main results for computational complexity are as follows.

1. **Global strategies.** We show that pushdown systems (one-player pushdown games) with mean-payoff objectives under global strategies can be solved in polynomial time, whereas solving pushdown games with mean-payoff objectives under global strategies is undecidable. We also consider pushdown systems with multidimensional mean-payoff objective, where the player has to achieve a non-negative mean-payoff value in every dimension. We show that this problem has a polynomial solution.

2. **Modular strategies.** Solving pushdown systems with single exit nodes with mean-payoff objectives under modular strategies is NP-hard, and pushdown games with mean-payoff objectives under modular strategies can be solved in NP. Thus both pushdown systems and pushdown games with mean-payoff objectives under modular strategies are NP-complete. We also consider multidimensional objectives and show that two-player games with a conjunctive condition are undecidable.

**Related publications:**

- Krishnendu Chatterjee, Yaron Velner: Mean-Payoff Pushdown Games. LICS 2012.

- Krishnendu Chatterjee, Yaron Velner: Hyperplane Separation Technique for Multidimensional Mean-Payoff Games. CONCUR 2013.

1.2.3 Applications

We show two applications for our results on multidimensional and infinite-state games. One application is in the field of theoretical game theory that considers games with payoff matrices (rather than a payoff function that is represented by a graph), and one application is in the field of static program analysis.
1.2.3.1 The complexity of infinitely repeated alternating move games

We consider infinite duration alternating move games. These games were previously studied by Roth, Balcan, Kalai and Mansour [85]. They presented an FPTAS for computing an approximate equilibrium, and conjectured that there is a polynomial algorithm for finding an exact equilibrium [84]. We extend their study in two directions: (1) We show that finding an exact equilibrium, even for two-player zero-sum games, is polynomial time equivalent to finding a winning strategy for a (two-player) mean-payoff game on graphs. The existence of a polynomial algorithm for the latter is a long standing open question in computer science. Our hardness result for two-player games suggests that two-player alternating move games are harder to solve than two-player simultaneous move games, while the work of Roth et al., suggests that for $k \geq 3$, $k$-player games are easier to analyze in the alternating move setting. (2) We show that optimal equilibria (with respect to the social welfare metric) can be obtained by pure strategies, and we present an FPTAS for computing a pure approximated equilibrium that is $\delta$-optimal with respect to the social welfare metric. This result extends the previous work by presenting an FPTAS that finds a much more desirable approximated equilibrium. We also show that if there is a polynomial algorithm for mean-payoff games on graphs, then there is a polynomial algorithm that computes an optimal exact equilibrium, and hence, (two-player) mean-payoff games on graphs are inter-reducible with $k$-player alternating move games, for any $k \geq 2$.


1.2.3.2 Quantitative Interprocedural Analysis

Static analysis techniques provide ways to obtain information about programs without actually running them on specific inputs. Static analysis explores the program behavior for all possible inputs and all possible executions. For non-trivial programs, it is impossible explore all the possibilities, and hence static analysis uses approximations (abstract interpretations) to account for all the possibilities [42]. Static analysis algorithms generally operate on the interprocedural control-flow graphs (for brevity ICFGs). An ICFG consists of a collection of control-flow graphs (CFGs), one for each procedure of the program. The CFG of each procedure has a unique entry node and a unique exit node, and other nodes represent statements of the program and conditions (in other words, basic blocks of the program). In addition, there are call and return nodes for each procedure which represent invoking of procedures and return from procedures. Call-transitions connect call nodes to entry nodes; and return-transitions connect exit nodes to return nodes. Algorithmic analysis of ICFGs provides the mathematical framework for static analysis of programs. Interprocedural analysis with objectives such as reachability, set-based information, etc have been extensively studied in the literature [22, 66, 74, 81–83].

In this work, together with Krishnendu Chatterjee and Andreas Pavlogiannis we
consider the quantitative analysis problem for interprocedural control-flow graphs. The input consists of an $ICFG$, a positive weight function that assigns every transition a positive integer-valued number, and a labelling of the transitions (events) as good, bad, and neutral events. The weight function assigns to each transition a numerical value that represents a measure of how good or bad an event is. The quantitative analysis problem asks whether there is a run of the $ICFG$ where the ratio of the sum of the numerical weights of the good events versus the sum of weights of the bad events in the long-run is at least a given threshold (or equivalently, to compute the maximal ratio among all valid paths in the $ICFG$). The quantitative analysis problem for $ICFG$s can be solved in polynomial time, and we present an efficient and practical algorithm for the problem. We show that several problems relevant for static program analysis, such as estimating the worst-case execution time of a program or the average energy consumption of a mobile application, can be modeled in our framework. We have implemented our algorithm as a tool in the Java Soot framework. We demonstrate the effectiveness of our approach with two case studies.

Related publication: Krishnendu Chatterjee, Andreas Pavlogiannis and Yaron Velner: Quantitative Interprocedural Analysis. POPL 2015.

1.3 Outline of Thesis and Related Publications

This thesis consists of three parts. In Part I we investigate multidimensional mean-payoff games on finite-graphs. In Chapter 3 we study two-player games with conjunctive condition, and in Chapters 4, 5 and 6 we study games with arbitrary conditions. In Chapter 4 we study the model-checking problem (one-player game), in Chapter 5 we study two-player games with the restriction that player 1 must play with finite-memory strategies, and in Chapter 6 we study two-player games with arbitrary strategy.

In Part II we investigate mean-payoff games over infinite-state pushdown graphs. In Chapter 7 we study pushdown games with global strategies, and in Chapter 8 we study pushdown games with modular strategies.

In Part III we consider two applications of our work. In Chapter 9 we study mean-payoff games with payoff matrices, and in Chapter 10 we apply our research on pushdown games to quantitative interprocedural analysis of programs.

Most of the contributions of this thesis have appeared in conference publications. We list the publications according to a chronicle order and the authors are presented in alphabetical order. The exact connection between the publications and the contributions is described in Section 1.2.

1. Krishnendu Chatterjee, Yaron Velner: Mean-Payoff Pushdown Games. LICS 2012.


ICALP 2013.


7. Yaron Velner: Robust Multidimensional Mean-Payoff Games are Undecidable. FOSSACS 2015.
Part I

Multidimensional Mean-Payoff Games on Finite Graphs
Chapter 2

Definitions and Outline

In this chapter we give the formal basic definitions and outline the structure of this part.

2.1 Definitions

Game graphs. A game graph $G = ((V, E), (V_1, V_2))$ consists of a finite directed graph $(V, E)$ with a finite set $V$ of $n$ vertices and a set $E$ of $m$ edges, and a partition $(V_1, V_2)$ of $V$ into two sets. The vertices in $V_1$ are player-1 vertices, where player 1 chooses the outgoing edges, and the vertices in $V_2$ are player-2 vertices, where player 2 (the adversary to player 1) chooses the outgoing edges. Intuitively game graphs are the same as AND-OR graphs. For a vertex $u \in V$, we write $\text{Out}(u) = \{v \in V \mid (u, v) \in E\}$ for the set of successor vertices of $u$. We assume that every vertex has at least one outgoing edge, i.e., $\text{Out}(u)$ is non-empty for all vertices $u \in V$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial vertex, and then they take moves indefinitely in the following way. If the token is on a vertex in $V_1$, then player 1 moves the token along one of the edges going out of the vertex. If the token is on a vertex in $V_2$, then player 2 does likewise. The result is an infinite path in the game graph, called plays. Formally, a play is an infinite sequence $\pi = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that $(v_j, v_{j+1}) \in E$ for all $j \geq 0$.

A game graph is called a one-player game if either $V_1 = \emptyset$ or $V_2 = \emptyset$ (i.e., only one player makes the decisions).

Strategies. A strategy for a player is a rule that specifies how to extend plays. Formally, a strategy $\tau$ for player 1 is a function $\tau: V^* \cdot V_1 \rightarrow V$ that, given a finite sequence of vertices (representing the history of the play so far) which ends in a player 1 vertex, chooses the next vertex. The strategy must choose only available successors, i.e., for all $w \in V^*$ and $v \in V_1$ we have $\tau(w \cdot v) \in \text{Out}(v)$. The strategies for player 2 are defined analogously. A strategy is memoryless if it is independent of the history and only depends on the current vertex. Formally, a memoryless strategy for player 1 is a function $\tau: V_1 \rightarrow V$ such that $\tau(v) \in \text{Out}(v)$ for all $v \in V_1$, and analogously for player 2 strategies. Given a starting vertex $v \in V$, a strategy $\tau$ for player 1, and a strategy $\sigma$ for player 2, there is a unique play, denoted $\pi(v, \tau, \sigma) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined
as follows: $v_0 = v$ and for all $j \geq 0$, if $v_j \in V_1$, then $\tau((v_0, v_1, \ldots, v_j)) = v_{j+1}$, and if $v_j \in V_2$, then $\sigma((v_0, v_1, \ldots, v_j)) = v_{j+1}$. A play $\pi'$ is consistent with player-1 strategy $\tau$ from an initial vertex $v$ if there exists a player-2 strategy such that $\pi' = \pi(v, \tau, \sigma)$.

A strategy is called a finite-memory strategy if it can be implemented by a finite-state machine. We formally define finite-memory strategies in Chapter 5.

**Multidimensional mean-payoff objectives.** For multidimensional mean-payoff objectives we will consider game graphs along with a weight function $w : E \to \mathbb{Z}^k$ that maps each edge to a vector of integer weights. We denote by $W$ the maximal absolute value of the weights. For a finite path $\pi$, we denote by $w(\pi)$ the sum of the weight vectors of the edges in $\pi$ and $\text{Avg}(\pi) = \frac{w(\pi)}{|\pi|}$, where $|\pi|$ is the length of $\pi$, denotes the average vector of the weights. We denote by $\text{Avg}_i(\pi)$ the projection of $\text{Avg}(\pi)$ to the $i$-th dimension. For an infinite path $\pi$, let $\pi_t$ denote the finite prefix of length $t$ of $\pi$; and we define $\text{LimInfAvg}_i(\pi) = \liminf_{t \to \infty} \text{Avg}_i(\pi_t)$ and analogously $\text{LimSupAvg}_i(\pi)$ with lim inf replaced by lim sup. For an infinite path $\pi$, we denote by $\text{LimInfAvg}(\pi) = (\text{LimInfAvg}_1(\pi), \ldots, \text{LimInfAvg}_k(\pi))$ (resp. $\text{LimSupAvg}(\pi) = (\text{LimSupAvg}_1(\pi), \ldots, \text{LimSupAvg}_k(\pi))$) the limit-inf (resp. limit-sup) vector of the averages (long-run average or mean-payoff objectives). The mean-payoff vector is the $2k$-dimensional vector $(\text{LimInfAvg}(\pi), \text{LimSupAvg}(\pi))$.

A boolean multidimensional objective is obtained by a boolean formula over the atoms $(x_i \sim \nu_i)$, where $x_i$ is either $\text{LimInfAvg}_i$ or $\text{LimSupAvg}_i$ (for $i = 1, \ldots, k$), $\sim \in \{\geq, >, \leq, <\}$, and $\nu_i \in \mathbb{Q}$ is a threshold. For example,

$$\varphi = (\text{LimInfAvg}_1 > 8 \land \text{LimSupAvg}_2 \leq 11) \lor \neg(\text{LimInfAvg}_3 \geq 0 \land \text{LimSupAvg}_4 < 0)$$

is a possible condition. The boolean value that $\varphi$ assigns to an infinite path $\pi$ is

$$\varphi(\pi) = (\text{LimInfAvg}_1(\pi) > 8 \land \text{LimSupAvg}_2(\pi) \leq 11) \lor \neg(\text{LimInfAvg}_3(\pi) \geq 0 \land \text{LimSupAvg}_4(\pi) < 0)$$

The next remark shows that it is enough to consider only zero thresholds (i.e, $\nu_i = 0$).

**Remark 1.** A mean-payoff objective is invariant to the shift operation, i.e., if in a dimension $i$, we require that the mean-payoff is at least $\nu_i$, then we simply subtract $\nu_i$ in the weight vector from every edge in the $i$-th dimension and require the mean-payoff is at least $0$ in dimension $i$. Hence the comparison with threshold zero is without loss of generality.

A quantitative multidimensional objective is a mean-payoff expression that is obtained by the algebraic operations of min, max, sum and numerical complement (multiplication by $-1$) over the atoms $x_i$, where $x_i$ is either $\text{LimInfAvg}_i$ or $\text{LimSupAvg}_i$ (for $i = 1, \ldots, k$). A possible expression is

$$E = \min(\text{LimInfAvg}_1, -\text{LimSupAvg}_2) + \max(\text{LimInfAvg}_3 - \text{LimSupAvg}_4, \text{LimSupAvg}_5)$$
and the (one-dimensional) value that $E$ assigns to an infinite path $\pi$ is

$$\min(\text{LimInfAvg}_1(\pi), -\text{LimSupAvg}_2(\pi)) + \max(\text{LimInfAvg}_3(\pi) - \text{LimSupAvg}_4(\pi), \text{LimSupAvg}_5(\pi))$$

The class of mean-payoff expressions was introduced in [32] and it generalizes the class of boolean objectives that consists only of weak inequalities, as, for example, the boolean formula $(x_1 \geq 0 \land x_2 \leq 0) \lor (x_3 \geq 0)$ is equivalent to $\max(\min(x_1, -x_2), x_3) \geq 0$. Hence, in most of this part, we consider the class mean-payoff expressions as the canonical form that represents multidimensional objectives.

**Conjunctive objectives.** An interesting special case is the conjunctive condition, namely $\varphi = \bigwedge_{i=1}^k (x_i \geq 0)$, and in the quantitative version $E = \min(x_1, \ldots, x_k)$, for $x_i \in \{\text{LimInfAvg}_i, \text{LimSupAvg}_i\}$.

**Winning strategies and optimal strategies.** For a boolean condition $\varphi$, a player-1 strategy $\tau$ is a winning strategy from a set $U$ of vertices, if for all player-2 strategies $\sigma$ and all $v \in U$ the path $\pi(v, \tau, \sigma)$ satisfies $\varphi$. A player-2 strategy is a winning strategy from a set $U$ of vertices if for all player-1 strategies $\tau$ and for all $v \in U$ we have that the path $\pi(v, \tau, \sigma)$ does not satisfy $\varphi$.

For a quantitative objective $E$, we say that a player-1 strategy $\tau$ achieves a value $\nu$ if the strategy is winning for the boolean objective $E \geq \nu$. A strategy $\tau$ is optimal if there exists a threshold $\nu$ such that $\tau$ achieves $\nu$, and there is no player-1 strategy that achieves a (strictly) greater value. Similarly, for an $\epsilon > 0$, a strategy $\tau$ is $\epsilon$-optimal if there exists a threshold $\nu$ such that $\tau$ achieves $\nu$, and there is no player-1 strategy that achieves a value $\nu + \epsilon$.

**2.2 Outline**

In the next chapter we consider games with conjunctive objectives and introduce a hyperplane separation technique that reduce the problem to a single-dimensional mean-payoff game and yields a pseudo-polynomial algorithm for games with fixed number of dimensions. In the rest of the chapters we consider general mean-payoff expressions. In Chapter 4, we investigate the model-checking problems (one-player games); in Chapter 5, we study two-player games with the restriction that player 1 must play with finite-memory strategy; and in Chapter 6 we study two-player games with arbitrary strategies.
Chapter 3

Hyperplane Separation Technique for Conjunctive Multidimensional Mean-Payoff Games

In this chapter we present a hyperplane based technique that yields a pseudo-polynomial algorithm for (conjunctive) multidimensional mean-payoff objectives. We first present an overview of the previous results, followed by the open problems, and finally our contributions.

Previous results and open questions. Finite-state games with a one-dimensional mean-payoff objective can be decided in NP ∩ coNP [47, 110], and pseudo-polynomial time algorithms exist for mean-payoff games [26, 110]: the current fastest known algorithm works in time $O(n \cdot m \cdot W)$, where $n$ is the number of vertices, $m$ is the number of edges, and $W$ is the maximal absolute value of the weights [26]. Finite-state games with (conjunctive) multidimensional mean-payoff objectives are coNP-complete with weights in $\{-1, 0, 1\}$ (i.e., the weights are bounded by a constant) but with arbitrary dimensions [35], and the current best known algorithm works in time $O(2^n \cdot \text{poly}(n, m, \log W))$.

The next question has remained open for analysis of finite-state games with multidimensional mean-payoff objectives where the goal of player 1 is to ensure that the mean-payoff is at least zero in all dimensions: Can finite-state game graphs with multidimensional mean-payoff objectives with 2 or 3 dimensions and constant weights be solved in polynomial time? (note that with arbitrary dimensions the problem is coNP-complete, and for arbitrary weights no polynomial time algorithm is known even for the one-dimensional case). The above question is not only of theoretical interest, but stem from practically motivated problems of formal analysis of reactive systems.

Our contributions. In this chapter we present a hyperplane separation technique and we use it to answer the question above. We present an algorithm for finite-state games with multidimensional mean-payoff objectives of $k$-dimensions that works in time $O(n^2 \cdot m \cdot k \cdot W \cdot (k \cdot n \cdot W)^{k^2+2k+1})$, and thus for constant weights and any constant $k$ (not only $k = 2$ or $k = 3$) our algorithm is polynomial.

Given our polynomial-time algorithms when the parameters are fixed for finite-state multidimensional mean-payoff games, a natural question is whether they are fixed parameter tractable, e.g., could we obtain an algorithm that runs in time $f(k) \cdot O(\text{poly}(n, m, W))$ for finite-state multidimensional mean-payoff games, for some
computable function $f$ (e.g., exponential or double exponential). We show the hardness of fixed parameter tractability problem by reducing the long-standing open problem of fixed parameter tractability of parity games to multidimensional mean-payoff games, i.e., fixed parameter tractability of any of the above problems would imply fixed parameter tractability of parity games.

**Bibliographic note.** The results of this chapter were first published in: Krishnendu Chatterjee, Yaron Velner: Hyperplane Separation Technique for Multidimensional Mean-Payoff Games. CONCUR 2013.

### 3.1 Definitions

In this Chapter, we consider only conjunction of $\text{LimInfAvg}_i$ atoms. In [98] we showed a polynomial reduction from solving a games with a condition $\bigwedge_{i=1}^{\ell} \text{LimInfAvg}_i \geq 0 \land \bigwedge_{i=\ell+1}^{k} \text{LimSupAvg}_i$ to $k-\ell$ games with a condition $\bigwedge_{i=1}^{\ell+1} \text{LimInfAvg}_i \geq 0$ with the same weights. Hence, a (pseudo) polynomial solution of conjunction of lim-inf atoms implies a (pseudo) polynomial solution for an arbitrary conjunction. As we consider only the $\text{LimInfAvg}$ metric, we abbreviate $\text{LimInfAvg}$ with $\text{LimAvg}$.

**Graphs obtained under memoryless strategies.** A player-1 graph is a special case of a game graph where all vertices in $V_2$ have a unique successor (and player-2 graphs are defined analogously). Given a memoryless strategy $\sigma$ for player 2, we denote by $G^\sigma$ the player-1 graph obtained by removing from all player-2 vertices the edges that are not chosen by $\sigma$.

**Winning strategies.** A player-1 strategy $\tau$ is a winning strategy from a set $U$ of vertices, if for all player-2 strategies $\sigma$ and all $v \in U$ we have $\text{LimAvg}(\pi(v, \tau, \sigma)) \geq \vec{0}$. A player-2 strategy is a winning strategy from a set $U$ of vertices if for all player-1 strategies $\tau$ and for all $v \in U$ we have that the path $\pi(v, \tau, \sigma)$ does not satisfy $\text{LimAvg}(\pi(v, \tau, \sigma)) \geq \vec{0}$. The winning region for a player is the largest set $U$ such that the player has a winning strategy from $U$.

### 3.2 Hyperplane separation algorithm

In this section we present our algorithm to decide the existence of a winning strategy for player 1 in finite-state multidimensional mean-payoff games.

**Hyperplane separation technique.** Our key insight is to search for a hyperplane $\mathcal{H}$ such that player 2 can ensure a mean-payoff vector that lies below $\mathcal{H}$. Intuitively, we show that if such a hyperplane exists, then any point in space that is below $\mathcal{H}$ is negative in at least one dimension, and thus the multidimensional mean-payoff objective for player 1 is violated. Conversely, we show that if for all hyperplanes $\mathcal{H}$ player 1 can achieve a mean-payoff vector that lies above $\mathcal{H}$, then player 1 can ensure the multidimensional mean-payoff objective. The technical argument relies on the fact that if we have an infinite sequence of unit vectors $\vec{b}_1, \vec{b}_2, \ldots$ and $\vec{b}_\ell$ lies above the hyperplane that is normal to $\sum_{j=1}^{\ell-1} \vec{b}_j$, then $\liminf_{\ell \to \infty} \frac{1}{\ell} \cdot \sum_{j=1}^{\ell} \vec{b}_j = \vec{0}$.

**Multiple dimensions to one dimension.** Given a multidimensional weight function $w$
CHAPTER 3. HYPERPLANE SEPARATION TECHNIQUE FOR CONJUNCTIVE MULTIDIMENSIONAL MEAN-PAYOFF GAMES

$$(1, -2) \xrightarrow{(0, 0)} (0, 0) \xrightarrow{(0, 0)} (-2, 1)$$

$$(-3, -3)$$

Figure 3.1: Game graph $G_1$.

$$(2, -1) \xrightarrow{(0, 0)} (0, 0) \xrightarrow{(0, 0)} (-1, 2)$$

$$(6, -3)$$

Figure 3.2: Game graph $G_2$.

and a vector $\vec{\lambda}$, we denote by $w \cdot \vec{\lambda}$ the one-dimensional weight function that assigns every edge $e$ the weight value $w(e)^T \cdot \vec{\lambda}$, where $w(e)^T$ is the transpose of the weight vector $w(e)$. We show that with the hyperplane technique we can reduce a game with multidimensional mean-payoff objective to the same game with a one-dimensional mean-payoff objective. A vector $\vec{b}$ lies above a hyperplane $H$ if $\vec{\lambda}$ is the normal vector of $H$ and $\vec{b}^T \cdot \vec{\lambda} \geq 0$. Hence, player 1 can achieve a mean-payoff vector that lies above $H$ if and only if player 1 can ensure the one-dimensional mean-payoff objective with weight function $w(e) \cdot \vec{\lambda}$.

Examples. Consider the game graph $G_1$ (Figure 3.1) where all vertices belong to player 1. The weight function $w_1$ labels each edge with a two-dimensional weight vector. In $G_1$, player 1 can ensure all mean-payoff vectors that are convex combination of $(1, -2), (-2, 1)$ and $(-1, -1)$ (see Figure 3.3). By Figure 3.3, all the vectors reside below the hyperplane $y = -x$, and consider the normal vector $\vec{\lambda} = (1, 1)$ to the hyperplane $y = -x$. All the cycles in $G_1$ with weight function $w_1 \cdot \vec{\lambda}$ (shown in Figure 3.5) have negative weights. Therefore player 1 loses in the one-dimensional mean-payoff objective.

Consider the game graph $G_2$ (Figure 3.2) with all player-1 vertices; where player 1 can achieve any mean-payoff vector that is a convex combination of $(2, -1), (-1, 2)$ and $(-2, -1)$ (see Figure 3.4). By Figure 3.4, every two-dimensional hyperplane that passes through the origin intersects with the feasible region. Thus, no separating hyperplane exists.

Basic lemmas and assumptions. We now prove two lemmas to formalize the intuition related to reduction to one-dimensional mean-payoff games. Lemma 1 requires two assumptions, which we later show (in Lemma 4) how to deal with. The assumptions are as follows: (1) The first assumption (we refer as Assumption 1) is that every outgoing edge of player-2 vertices is to a player-1 vertex; formally, $E \cap (V_2 \times V) \subseteq E \cap (V_2 \times V_1)$.

$$-1 \xrightarrow{0} 0 \xrightarrow{0} -1$$

Figure 3.5: Game graph $G_1$ with weight function $\vec{\lambda} \cdot w_1$ for $\vec{\lambda} = (1, 1)$. 
The second assumption (we refer as Assumption 2) is that every player-1 vertex has \( k \) self-loop edges \( e_1, \ldots, e_k \) such that \( w_i(e_j) = 0 \) if \( i \neq j \) and \( w_i(e_i) = -1 \). Let us denote by \( \text{Win}^2_\lambda \) the player-2 winning region in the multidimensional mean-payoff game with weight function \( w \), and by \( \text{Win}^2_\lambda \) the player-2 winning region in the one-dimensional mean-payoff game with the weight function \( w \cdot \vec{\lambda} \). The next lemma shows that if \( \text{Win}^2_\lambda \neq \emptyset \), then \( \text{Win}^2 \neq \emptyset \); i.e., presents a sufficient condition for the non-emptiness of \( \text{Win}^2 \).

**Lemma 1.** Given a game graph \( G \) that satisfies Assumption 1 and Assumption 2, and a multidimensional mean-payoff objective with weight function \( w \), for every \( \vec{\lambda} \in \mathbb{R}^k \) we have \( \text{Win}^2_{\vec{\lambda}} \subseteq \text{Win}^2 \); (hence, if \( \text{Win}^2_{\vec{\lambda}} \neq \emptyset \), then \( \text{Win}^2 \neq \emptyset \)).

**Proof.** Let \( \sigma \) be a player-2 winning strategy in \( G \) from an initial vertex \( v_0 \) (i.e., winning strategy from the set \( \{v_0\} \)) for the one-dimensional mean-payoff objective with weight function \( w \cdot \vec{\lambda} \). We first observe that we must have \( \vec{\lambda} \in (0, \infty)^k \); otherwise if \( \lambda_i \in (-\infty, 0] \) then by Assumption 1 the weight of the \( i \)-th self-loop of a player-1 vertex would be non-negative, and player 1 can ensure the mean-payoff objective from all vertices (by Assumption 2 all plays arrive to a player-1 vertex within one step), contradicting \( v_0 \) is winning for player 2. We claim that \( \sigma \) is also a player-2 winning strategy with respect to the multidimensional mean-payoff objective. Indeed, let \( \rho \) be a play that is consistent with \( \sigma \). Since \( \sigma \) is a player-2 winning strategy for the mean-payoff objective with weight function \( w \cdot \vec{\lambda} \), it follows that there exists a constant \( c > 0 \) such that there are infinitely many prefixes of \( \rho \) with average weight (according to \( w \cdot \vec{\lambda} \)) at most \( -c \). Let \( \lambda_{\text{min}} = \min\{\lambda_i \mid 1 \leq i \leq k\} \) be the minimum value of \( \vec{\lambda} \) among its dimension. Since \( \vec{\lambda} \in (0, \infty)^k \), it follows that \( \lambda_{\text{min}} > 0 \). Since there are finitely many dimensions there must be a dimension \( i \) for which there are infinitely many prefixes of \( \rho \) with average weight at most \( -c \frac{\lambda_{\text{min}}}{k} < 0 \) in dimension \( i \). Hence, the mean-payoff value of dimension \( i \) is negative, and thus the multidimensional mean-payoff objective is violated. Hence \( \sigma \) is a player-2 winning strategy from \( v_0 \) against the multidimensional mean-payoff objective. \( \square \)

We now present a lemma that will complement Lemma 1, and the following lemma does not require Assumption 1 or Assumption 2.

**Lemma 2.** Given a game graph \( G \) and a multidimensional mean-payoff objective with weight function \( w \), if for all \( \vec{\lambda} \in \mathbb{R}^k \) we have \( \text{Win}^2_{\vec{\lambda}} = \emptyset \), then we have \( \text{Win}^2 = \emptyset \).

**Proof.** Since \( \text{Win}^2_{\vec{\lambda}} = \emptyset \) for every \( \vec{\lambda} \in \mathbb{R}^k \), it follows by the determinacy of one-dimensional mean-payoff games [47] that for all \( \vec{\lambda} \in \mathbb{R}^k \), player 1 can ensure the one-dimensional mean-payoff objective with weight function \( w \cdot \vec{\lambda} \) in \( G \) (from all initial vertices). We now present an explicit construction of a player-1 winning strategy for the multidimensional mean-payoff objective in \( G \). For a vector \( \vec{\lambda} \in \mathbb{R}^k \), let \( \tau_{\vec{\lambda}} \) be a memoryless player-1 winning strategy in \( G \) from all vertices for the one-dimensional mean-payoff game with weight function \( w \cdot \vec{\lambda} \) (note that uniform memoryless winning strategies that
ensure winning from all vertices in the winning region exist in one-dimensional mean-payoff games by the results of [47]). We construct a player-1 winning strategy $\tau$ for the multidimensional objective in the following way:

- Initially, set $\vec{b}_0 := (1, 1, \ldots, 1)$.
- For $i = 1, 2, \ldots, \infty$, in iteration $i$ play as follows:
  - Set $\vec{\lambda}_i := -\vec{b}_{i-1}$. In $\tau$, player 1 plays according to $\tau_{\vec{\lambda}_i}$ for $i$ rounds.
  - Let $\rho_i$ be the play suffix that was formed in the last $i$ rounds (or steps) of the play. From $\rho_i$ we obtain the part of $\rho_i$ that consists of cycles (that are possibly repeated) and denote the part as $\rho_i^2$; and an acyclic part $\rho_i^1$ of length at most $n$. Informally, $\rho_i^2$ consists of cycles that appear in $\rho_i$, and if a cycle $C$ is repeated $j$ times in $\rho_i$ then it is included $j$ times in $\rho_i^2$; see Figure 3.6 for an illustration. We note that the decomposition to $\rho_i^1$ and $\rho_i^2$ is unique.
  - Set $\vec{b}_i := \vec{b}_{i-1} + w(\rho_i^2)$; and proceed to the next iteration.

In order to prove that $\tau$ is a winning strategy, it is enough to prove that for every play $\rho$ that is consistent with $\tau$, the Euclidean norm of the average weight vector tends to zero as the length of the play tends to infinity.

We first compute the Euclidean norm of $\vec{b}_i$. For this purpose we observe that $\tau_{\vec{\lambda}_i}$ is a memoryless winning strategy for the one-dimensional mean-payoff game with weight function $w \cdot \vec{\lambda}_i$; and hence it follows that for every cycle $C$ in the graph $G^T_{\vec{\lambda}_i}$ the sum of the weights of $C$ according to $w \cdot \vec{\lambda}_i$ is non-negative. Since $\rho_i^2$ is composed of cyclic paths, we must have $w(\rho_i^2)^T \cdot \vec{\lambda}_i \geq 0$; and hence, we have $w(\rho_i^2)^T \cdot \vec{b}_{i-1} \leq 0$. Thus we get that

$$|\vec{b}_i| = |\vec{b}_{i-1} + w(\rho_i^2)| = \sqrt{|\vec{b}_{i-1}|^2 + 2 \cdot w(\rho_i^2)^T \cdot \vec{b}_{i-1} + |w(\rho_i^2)|^2} \leq \sqrt{|\vec{b}_{i-1}|^2 + |w(\rho_i^2)|^2}$$

Figure 3.6: The path segment $\rho_i$ is decomposed into the cycles (possibly repeated) as $\rho_i^2 = C_1 \cdot C_1 \cdot C_2 \cdot C_2 \cdot C_2 \cdot C_3$; and the acyclic part $\rho_i^1 = e_1 \cdot e_2 \cdot e_3 \cdot e_4 \cdot e_5 \cdot e_6$. 


Since \( W \) is the maximal absolute value of the weights, it follows that \( W \cdot \sqrt{k} \) is a bound on the Euclidean norm of any average weight vector. Since the length of \( \rho^2_i \) is at most \( i \) (it was a part of the suffix of last \( i \) rounds) we get that

\[
|\vec{b}_i| \leq \sqrt{|\vec{b}_{i-1}|^2 + k \cdot W^2 \cdot i^2}.
\]

By a simple induction we obtain that

\[
|\vec{b}_i| \leq \sqrt{k \cdot W^2 \cdot \sum_{j=1}^{i} j^2}.
\]

We are now ready to compute the the Euclidean norm of the play after the \( i \)-th iteration. We denote the weight vector after the \( i \)-th iteration by \( \vec{x}_i \) and observe that

\[
\vec{x}_i = \vec{b}_i + \sum_{j=1}^{i} w(\rho^1_j).
\]

By the Triangle inequality we get that

\[
|\vec{x}_i| \leq |\vec{b}_i| + \sum_{j=1}^{i} |w(\rho^1_j)|.
\]

Since the length of \( \rho^1_i \) is at most \( n \) and by the bound we obtained over \( \vec{b}_i \) we get that

\[
|\vec{x}_i| \leq \sqrt{k \cdot W^2 \cdot i^3 + i \cdot n \cdot W \cdot \sqrt{k}}.
\]

For a position \( j \) of the play between iteration \( i \) and iteration \( i+1 \), let us denote by \( \vec{y}_j \) the weight vector after the play prefix at position \( j \). Since there are \( i \) steps played in iteration \( i \) we have

\[
|\vec{y}_j| \leq |\vec{x}_i| + i \cdot W \cdot \sqrt{k}.
\]

Finally, since after the \((i-1)\)-th iteration \( \sum_{t=1}^{i-1} t = i \cdot (i-1)/2 \) rounds were played, we get that the Euclidean norm of the average weight vector, namely, \( \frac{1}{i} \sum_{j=1}^{i} |\vec{y}_j| \), tends to zero as \( i \) tends to infinity. Formally we have

\[
\lim_{j \to \infty} \frac{|\vec{y}_j|}{j} \leq \lim_{i \to \infty} \frac{\sqrt{k \cdot W^2 \cdot i^3 + i \cdot n \cdot W \cdot \sqrt{k} + i \cdot W \cdot \sqrt{k}}}{i \cdot (i-1)/2} = 0.
\]

It follows that the limit average of the weight vectors is zero and hence the desired result follows.
Notations. For the rest of this section, we denote $M = (k \cdot n \cdot W)^{k+1}$, where $W$ is the maximal absolute value of the weight function. For a positive integer $\ell$, we will denote by $\mathbb{Z}_\ell^+ = \{i \mid -\ell \leq i \leq \ell\}$ (resp. $\mathbb{Z}_\ell^- = \{i \mid 1 \leq i \leq \ell\}$) the set of integers (resp. positive integers) from $-\ell$ to $\ell$.

**Lemma 3.** Let $G$ be a game graph with a multidimensional mean-payoff objective with a weight function $w$. There exists $\tilde{x}_0 \in \mathbb{R}^k$ for which player-2 winning region is non-empty in $G$ for the one-dimensional mean-payoff objective with weight function $w \cdot \tilde{x}_0$ if and only if there exists $\tilde{\lambda} \in (\mathbb{Z}_M^\pm)^k$ such that the player-2 winning region is non-empty in $G$ for the one-dimensional mean-payoff objective with weight function $w \cdot \tilde{\lambda}$.

**Proof.** Suppose that player 2 has a memoryless winning strategy $\sigma$ in $G$ from an initial vertex $v_0$ for the one-dimensional mean-payoff objective with weight function $w \cdot \tilde{x}_0$. Let $C_1, \ldots, C_m$ be the simple cycles that are reachable from $v_0$ in the graph $G^\sigma$. Since $\sigma$ is a player-2 winning strategy it follows that $w(C_i)^T \cdot \tilde{x}_0 < 0$ for every $i \in \{1, \ldots, m\}$. We note that for all $1 \leq i \leq m$ we have $w(C_i) \in (\mathbb{Z}_n^\pm)^k$ (since $C_i$ is a simple cycle, in every dimension the sum of the weights is between $-n \cdot W$ and $n \cdot W$). Then by [77, Lemma 2, items c and d] it follows that there is a vector of integers $\tilde{\lambda}$ such that $w(C_i)^T \cdot \tilde{\lambda} \leq -1 < 0$, for all $1 \leq i \leq m$; and $\tilde{\lambda} \in (\mathbb{Z}_M^\pm)^k$. Since all the reachable cycles from $v_0$ in $G^\sigma$ are negative according to $w \cdot \tilde{\lambda}$, we get that $\sigma$ is a winning strategy for the one-dimensional mean-payoff game with weight function $w \cdot \tilde{\lambda}$; and hence the proof for the direction from left to right follows. The proof for the converse direction is trivial. 

The next lemma removes the two assumptions of Lemma 1.

**Lemma 4.** Let $G$ be a game graph with a multidimensional mean-payoff objective with a weight function $w$. The following assertions hold: (1) $\bigcup_{\tilde{x} \in (\mathbb{Z}_M^\pm)^k} \text{Win}_x^2 \subseteq \text{Win}^2$. (2) If $\bigcup_{\tilde{x} \in (\mathbb{Z}_M^\pm)^k} \text{Win}_x^2 = \emptyset$, then $\text{Win}^2 = \emptyset$.

**Proof.** We first show how to construct a game graph $\tilde{G}$ from $G$ that satisfies the two assumptions (Assumption 1 and Assumption 2) and has the same winning regions (for the multidimensional objective) as in $G$.

1. **(Assumption 1).** Given any game graph $G$ there exists a linear transformation to satisfy Assumption 1 by simply adding a dummy vertex for every outgoing edge of a player 2 vertex (i.e., for every edge $e = (u, v)$ with $u, v \in V_2$, we add a vertex $e$, edges $(u, e)$ with weight $w(e)$ and $(e, v)$ with weight $\tilde{0}$, and $e$ is a player-1 vertex with a single outgoing edge).

2. **(Assumption 2).** First, note that adding several self-loop edges creates a multi-graph, but a dummy player-2 vertex can be put for every such edge to ensure that we do not have a multi-graph. Second we observe that adding the self-loop edges of Assumption 2 do not affect winning for player 1, as if there is a winning strategy for player 1, then there is one that never chooses the self-loop edges of
Assumption 2 because the self-loop edges are non-positive in every dimension and negative in one dimension. 

For a game graph $G$ we denote by $\hat{G}$ the graph that is formed by the transformations above. We now establish the following claim:

**Claim.** The following two properties hold for the game graph $\hat{G}$: (i) if a vector $\vec{\lambda}$ is non-positive in (at least) one dimension, then player-2 winning region in $\hat{G}$ for the one-dimensional mean-payoff objective with weight function $w \cdot \vec{\lambda}$ is empty; and (ii) if a vector $\vec{\lambda}$ is positive in all dimensions, then player-2 winning region in $G$ and in $\hat{G}$ is the same for the one-dimensional mean-payoff objective with weight function $w \cdot \vec{\lambda}$. The first item of the claim holds due to the self-loops of Assumption 2, and Assumption 1 ensures that a player-1 vertex is reached within two steps (the same reasoning as used in Lemma 1). The second item of the claim holds because the weight of any simple cycle in $G$ is the same as in $\hat{G}$, and the weight of Assumption 2 self-loops are non-positive in every dimension and negative in one dimension (since $\vec{\lambda}$ is positive in all dimensions). Hence, a memoryless winning strategy in $G$ is also winning in $\hat{G}$ and vice-versa.

We now prove the two assertions of the lemma.

1. (First assertion). Consider that in $G$ we have $v \in \text{Win}_{\vec{\lambda}}^2$, for some vertex $v$ and a vector $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$. Then by the second item of the claim we get that $v \in \text{Win}_{\vec{\lambda}}^2$ also in $\hat{G}$, and then by Lemma 1 we get that $v \in \text{Win}_{\vec{\lambda}}^2$ (in $\hat{G}$). Finally, by the definition of the transformations, we get that player 2 is winning from $v$ for the multidimensional mean-payoff objective in $\hat{G}$ if and only if player 2 is winning from $v$ for the multidimensional mean-payoff objective in $G$. Thus $v \in \text{Win}_{\vec{\lambda}}^2$ in $G$ and the first assertion follows.

2. (Second assertion). For the second assertion consider that $\text{Win}_{\vec{\lambda}}^2 \neq \emptyset$ (in $G$) and we show that for some $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$ we have $\text{Win}_{\vec{\lambda}}^2 \neq \emptyset$ (in $G$). Suppose that $v \in \text{Win}_{\vec{\lambda}}^2$ for some vertex $v$ in $G$. Then by the definition of the transformation we have that $v \in \text{Win}_{\vec{\lambda}}^2$ also in $\hat{G}$. By Lemma 2 and Lemma 3 it follows that there is $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$ such that $v \in \text{Win}_{\vec{\lambda}}^2$ (in $\hat{G}$). By the first item of the claim we get that $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$. Finally, by the second item of the claim, and since $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$, we get that $v \in \text{Win}_{\vec{\lambda}}^2$ also in $G$, and thus the second assertion follows.

The desired result follows. 

To use the result of Lemma 4 iteratively to solve finite-state games with multidimensional mean-payoff objectives, we need the notion of attractors. For a set $U$ of vertices, $\text{Attr}_2(U)$ is defined inductively as follows: $U_0 = U$ and for all $i \geq 0$ we have $U_{i+1} = U_i \cup \{v \in V_1 \mid \text{Out}(v) \subseteq U_i\} \cup \{v \in V_2 \mid \text{Out}(v) \cap U_i \neq \emptyset\}$, and $\text{Attr}_2(U) = \bigcup_{i \geq 0} U_i$. Intuitively, from $U_{i+1}$ player 2 can ensure to reach $U_i$ in one step against all strategies of player 1, and thus $\text{Attr}_2(U)$ is the set of vertices such that player 2 can ensure to reach $U$ against all strategies of player 1 in finitely many steps. The set $\text{Attr}_2(U)$ can be computed in linear time [12, 57]. Observe that if $G$ is a game graph, then for all $U$,
the game graph induced by the set $V \setminus \text{Attr}_2(U)$ is also a game graph (i.e., all vertices in $V \setminus \text{Attr}_2(U)$ have outgoing edges in $V \setminus \text{Attr}_2(U)$). The following lemma shows that in multidimensional mean-payoff games, if $U$ is a set of vertices such that player 2 has a winning strategy from every vertex in $U$, then player 2 has a winning strategy from all vertices in $\text{Attr}_2(U)$, and we can recurse in the game graph after removal of $\text{Attr}_2(U)$.

**Lemma 5.** Consider a multidimensional mean-payoff game $G$ with weight function $w$. Let $U$ be a set of vertices such that from all vertices in $U$ there is a winning strategy for player 2. Then the following assertions hold: (1) From all vertices in $\text{Attr}_2(U)$ there is a winning strategy for player 2. (2) Let $Z$ be the set of vertices in the game graph induced after removal of $\text{Attr}_2(U)$ such that from all vertices in $Z$ player 2 has a winning strategy in the remaining game graph. Then from all vertices in $Z$, player 2 has a winning strategy in the original game graph.

**Proof.** The proof of the first item is as follows: from vertices in $\text{Attr}_2(U)$ first consider a strategy to ensure to reach $U$ (within finitely many steps), and once $U$ is reached switch to a winning strategy from vertices in $U$. The proof of second item is as follows: fix a winning strategy in the remaining game graph for vertices in $Z$ and a winning strategy from $\text{Attr}_2(U)$ for player 2. Consider any counter strategy for player 1. If $\text{Attr}_2(U)$ is ever reached, then the winning strategy from $\text{Attr}_2(U)$ ensures winning for player 2, and otherwise the winning strategy of the remaining game graph ensures winning.

**Algorithm.** We now present our iterative algorithm that is based on Lemma 4 and Lemma 5. In the current iteration $i$ of the game graph execute the following steps: sequentially iterate over vectors $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$; and if for some $\vec{\lambda}$ we obtain a non-empty set $U$ of winning vertices for player 2 for the one-dimensional mean-payoff objective with weight function $w \cdot \vec{\lambda}$ in the current game graph, remove $\text{Attr}_2(U)$ from the current game graph and proceed to iteration $i + 1$. Otherwise if for all $\vec{\lambda} \in (\mathbb{Z}_M^+)^k$, player 1 wins from all vertices for the one-dimensional mean-payoff objective with weight function $w \cdot \vec{\lambda}$, then the set of current vertices is the set of winning vertices for player 1. The correctness of the algorithm follows from Lemma 4 and Lemma 5.

**Complexity.** The algorithm has at most $n$ iterations, and each iteration solves at most $O(M^k)$ one-dimensional mean-payoff games. Thus the iterative algorithm requires to solve $O(n \cdot M^k)$ one-dimensional mean-payoff games with $m$ edges, $n$ vertices, and the maximal weight is at most $k \cdot W \cdot M$. Since one-dimensional mean-payoff games with $n$ vertices, $m$ edges, and maximal weight $W$ can be solved in time $O(n \cdot m \cdot W)$ [26], we obtain the following result.

**Theorem 1.** The set of winning vertices for player 1 in a multidimensional mean-payoff game with $n$ vertices, $m$ edges, $k$-dimensions, and maximal absolute weight $W$ can be computed in time $O(n^2 \cdot m \cdot k \cdot W \cdot (k \cdot n \cdot W)^{k^2+2\cdot k+1})$. 


3.3 Hardness for fixed parameter tractability

In this section we reduce finite-state parity games to finite-state multidimensional mean-payoff games with weights bounded linearly by the number of vertices. Note that our reduction is different from the standard reduction of parity games to one-dimensional mean-payoff games where exponential weights are necessary [58]. We start with the definition of parity games.

**Parity games.** A parity game consists of a finite-state game graph $G$ along with a priority function $p : E \rightarrow \{1, \ldots, k\}$ that maps every edge to a natural number (the priority). The objective of player 1 is to ensure that the *minimal* priority that occurs infinitely often in a play is even, and the goal of player 2 is the complement. The memoryless determinacy of parity games shows that for both players if there is a winning strategy, then there is a memoryless winning strategy [48].

**The reduction.** Given a game graph $G$ with $n$ vertices and with priority function $p : E \rightarrow \{1, \ldots, k\}$ we construct a multidimensional mean-payoff objective with weight function $w$ of $k$ dimensions on $G$ as follows: for every $i \in \{1, \ldots, k\}$ we assign $w_i(e)$ as follows:

- $0$ if $p(e) > i$;
- $-1$ if $p(e) \leq i$ and $p(e)$ is odd; and
- $n$ if $p(e) \leq i$ and $p(e)$ is even.

**Lemma 6.** From a vertex $v$, if player 1 wins the parity game, then she also wins the multidimensional mean-payoff game.

**Proof.** If player 1 is the winner in the parity game from $v$, then by memoryless determinacy of parity games there is memoryless winning strategy $\tau$. Since $\tau$ is winning in the parity game, then every simple cycle $C$ reachable from $v$ in $G^\tau$ is even (i.e., the minimum priority of $C$ is even). Given a cycle $C$ with minimum priority $i$ which is even we have (i) for $j < i$: $w_j(C) = 0$; and (ii) for $j \geq i$ there is at least one state with weight $n$, and the sum of all other weights is at least $-(n-1)$ (since there are at most $n$ edges of which one has weight $n$, and in the worst case all the remaining $n-1$ edges have weight $-1$); and hence $w_j(C) \geq 0$. Hence by the construction of the weight function it follows that the weight vector of $C$ is non-negative (in every dimension). Thus $\tau$ is a winning strategy for the multidimensional mean-payoff objective.

**Lemma 7.** From a vertex $v$, if player 2 wins the parity game, then she also wins the multidimensional mean-payoff game.

**Proof.** If player 2 is the winner in the parity game from $v$, then by memoryless determinacy she has a memoryless winning strategy $\sigma$. We claim that $\sigma$ is a winning strategy for player 2 in the multidimensional mean-payoff game. For this purpose we
CHAPTER 3. HYPERPLANE SEPARATION TECHNIQUE FOR CONJUNCTIVE
MULTIDIMENSIONAL MEAN-PAYOFF GAMES

first show that $\sigma$ is a winning strategy in the one-dimensional mean-payoff game with weight function $w \cdot \vec{\lambda}$, where $\ell = n^2$ and

$$\vec{\lambda} = (\ell^{k-1}, \ell^{k-2}, \ldots, \ell^{k-i}, \ldots, \ell^0)$$

Let $C$ be a simple cycle reachable from $v$ in the player-1 graph $G_\sigma$. Let $i$ be the minimal priority that occurs in $C$, and since $\sigma$ is winning for player 2, it follows that $i$ is odd. By the construction of the weight function we get that (i) $w_i(C) \leq -1$; (ii) for $j > i$: $w_j(C) \leq n^2 - 1 = \ell - 1$ (at least one edge has negative weight, and all other edges have weight at most $n$); and (iii) for $j < i$: $w_j(C) = 0$. Hence we get that

$$w(C)^T \cdot \vec{\lambda} \leq \ell^{k-i} + (\ell - 1) \cdot \sum_{j=i+1}^{k} \ell^{k-j} \leq \ell^{k-i} + (\ell - 1) \cdot \ell^{k-i-1} < 0$$

Hence, we get that every cycle reachable from $v$ in $G_\sigma$ is negative according to $w \cdot \vec{\lambda}$; and hence $\sigma$ is a winning strategy in the one-dimensional mean-payoff game for weight function $w \cdot \vec{\lambda}$. By Lemma 4 it follows that player 2 also wins in the multidimensional mean-payoff game from $v$.

**Theorem 2.** Let $G$ be a game graph with a parity objective defined by a priority function of $k$-priorities. We can construct in linear time a $k$-dimensional weight function $w$, with maximal weight $W$ bounded by $n$, such that a vertex is winning for player 1 in the parity game iff the vertex is winning for player 1 in the multidimensional mean-payoff game.

**Remark 2.** There exists a deterministic sub-exponential time algorithm for parity games [59] and also algorithms that run in time $O(n^{k/3} \cdot m)$ [86]; however obtaining a fixed parameter tractable algorithm for parity games that runs in time $O(f(k) \cdot \text{poly}(n,m))$ for any function $f$ (e.g., exponential, double exponential or even non-recursive) is a long-standing open problem. Our reduction (Theorem 2) shows that obtaining a fixed parameter tractable algorithm for multidimensional mean-payoff games that runs in time $O(f(k) \cdot \text{poly}(n,m,W))$ is not possible without first solving the fixed parameter tractability of parity games. We also point out that the hardness result does not hold for (conjunctive) multidimensional LimSupAvg-objectives, as if the weights are fixed, the problem can be solved in polynomial time [98].
Chapter 4

The Complexity of Model-Checking Problems with Mean-Payoff Expression Specifications.

In algorithmic verification of reactive systems, the system is modeled as a finite-state transition system, and requirements are captured as languages of infinite words over system observations [79, 80]. The classical verification framework only captures qualitative aspects of system behavior, and in order to describe quantitative aspects, for example, consumption of resources such as CPU and energy, the framework of quantitative languages was proposed [34].

Quantitative languages are a natural generalization of Boolean languages that assign to every word a real number instead of a Boolean value. With such languages, quantitative specifications can be formalized. In this model, an implementation $L_A$ satisfies (or refines) a specification $L_B$ if $L_A(\alpha) \leq L_B(\alpha)$ for all words $\alpha$.

This notion of refinement is a quantitative generalization of language inclusion, and it can be used to check for example if for each behavior, the long-run average response time of the system lies below the specified average response requirement. The other classical model-checking problems such as emptiness, universality, and language equivalence have also a natural quantitative extension. For example, the quantitative emptiness problem asks, given a quantitative language $L$ and a rational threshold $\nu$, whether there exists some word $\alpha$ such that $L(\alpha) \geq \nu$, and the quantitative universality problem asks whether $L(\alpha) \geq \nu$ for all words $\alpha$. We also consider the notion of distance between two quantitative languages $L_A$ and $L_B$, defined as $\sup_{\alpha \in \Sigma^\omega} |L_A(\alpha) - L_B(\alpha)|$.

In this work, we study the computational complexity of the classical model-checking problems for the class of quantitative languages that are defined by mean-payoff expression. In this chapter, we use a slightly more general definition for the class of mean-payoff expressions. We obtain an expression from $k$ (single-dimensional) weighted automata, rather than from a $k$-dimensional weighted graph (as defined in Chapter 2). This model is more general, as a $k$ dimensional graph can be represented by the product of $k$ weighted automata, and it allows a more compact representation of specifications. An expression is either a deterministic\(^1\) mean-payoff automaton, or it is the max, min or sum of two mean-payoff expressions. This class, introduced in [32], is robust as it is

\(^1\)We note that the restriction to deterministic automata is inherent; for nondeterministic automata all the decision problems are undecidable [31].
closed under the max, min, sum and the numerical complement operators [31].

The decidability of the classical decision problems, as well as the computability of the distance problem, was first established in [31]; in this chapter we describe alternative proofs for these results. Our proofs offer the following advantages: First, the proofs yield PSPACE complexity upper bounds, which match corresponding PSPACE lower bounds; in comparison to 4EXPTIME upper bounds achieved in [31]. Second, our proofs reside only in the frameworks of graph theory and basic linear-programing, which are common practices among the automata-theoretic community, whereas a substantial part of the proofs in [31] resides in the framework of computational geometry.

Our proofs are based on a reduction from the emptiness problem to the feasibility problem for a set of linear inequalities; for this purpose, inspired by the proofs in [98], we establish a connection between the emptiness problem and the problem of finding a multi-set of cycles, with certain properties, in a directed graph. The reduction also reveals how to compute the maximum value of an expression, and therefore the decidability of all mentioned problems is followed almost immediately.

This chapter is organized as follows: In the next section we formally define the class of mean-payoff expressions; in Section 4.2 we describe a PSPACE algorithm that computes the maximum value of an expression that does not contain the max operator; in Section 4.3 we show PSPACE algorithm for all the classical problems and in Section 4.4 we prove their corresponding PSPACE lower bounds.

Bibliographic note. The results of this chapter were first published in: Yaron Velner: The Complexity of Mean-Payoff Automaton Expression. ICALP 2012.

4.1 Mean-Payoff Automaton Expression

In this section we present the definitions of mean-payoff expressions from [31].

Quantitative languages. A quantitative language $L$ over a finite alphabet $\Sigma$ is a function $L : \Sigma^\omega \to \mathbb{R}$. Given two quantitative languages $L_1$ and $L_2$ over $\Sigma$, we denote by $\max(L_1, L_2)$ (resp., $\min(L_1, L_2)$, $\sum(L_1, L_2)$ and $-L_1$) the quantitative language that assigns $\max(L_1(\alpha), L_2(\alpha))$ (resp., $\min(L_1(\alpha), L_2(\alpha))$, $L_1(\alpha) + L_2(\alpha)$, and $-L_1(\alpha)$) to each word $\alpha \in \Sigma^\omega$. The quantitative language $-L$ is called the complement of $L$.

Cut-point languages. Let $L$ be a quantitative language over $\Sigma$. Given a threshold $\nu \in \mathbb{R}$, the cut-point language defined by $(L, \nu)$ is the language $L^{\geq \nu} = \{ \alpha \in \Sigma^\omega | L(\alpha) \geq \nu \}$.

Weighted automata. A (deterministic) weighted automaton is a tuple $A = \langle Q, q_I, \Sigma, \delta, w \rangle$, where (i) $Q$ is a finite set of states, $q_I \in Q$ is the initial state, and $\Sigma$ is a finite alphabet; (ii) $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions such that for every $q \in Q$ and $\sigma \in \Sigma$ the size of the set $\{ q' \in Q | (q, \sigma, q') \in \delta \}$ is exactly 1; and (iii) $w : \delta \to \mathbb{Q}$ is a weight function, where $\mathbb{Q}$ is the set of rationals.

The product of weighted automata. The product of the weighted automata $A_1, \ldots, A_n$ such that $A_i = \langle Q_i, q^i_I, \Sigma, \delta_i, w_i \rangle$ is the multidimensional weighted au-
tomaton $A = A_1 \times \cdots \times A_n = \langle Q_1 \times \cdots \times Q_n, (q^1_1, \ldots, q^n_1), \Sigma, \delta, w \rangle$ such that $t = ((q_1, \ldots, q_n), \sigma, (q'_1, \ldots, q'_n)) \in \delta$ if $t_i = (q_i, \sigma, q'_i) \in \delta_i$ for all $i \in \{1, \ldots, n\}$, and $w(t) = (w_1(t_1), \ldots, w_n(t_n)) \in \mathbb{Q}^n$. We denote by $A_i$ the projection of the automaton $A$ to dimension $i$.

**Words and runs.** A word $\alpha \in \Sigma^\omega$ is an infinite sequence of letters from $\Sigma$. A run of a weighted automaton $A$ over an infinite word $\alpha = \sigma_1\sigma_2\ldots$ is the (unique) infinite sequence $r = q_0\sigma_1q_1\sigma_2\ldots$ of states and letters such that $q_0 = q_1$, and $(q_i, \sigma_{i+1}, q_{i+1}) \in \delta$ for all $i \geq 0$. We denote by $w(\alpha) = w(r) = v_0v_1\ldots$ the sequence of weights that occur in $r$ where $v_i = w(q_i, \sigma_{i+1}, q_{i+1})$ for all $i \geq 0$.

**Quantitative language of mean-payoff automata.** The mean-payoff value (or limit average) of a sequence $\pi = v_0v_1\ldots$ of real numbers is either $\text{LimInfAvg}(\pi) = \lim\inf_{n \to \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$; or $\text{LimSupAvg}(\pi) = \lim\sup_{n \to \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$. The quantitative language $\mathcal{A}$ of a weighted automaton $A$ is defined by $\mathcal{A}(\alpha) = \text{LimInfAvg}(w(\alpha))$; analogously the quantitative language $\overline{\mathcal{A}}$ is defined by $\overline{\mathcal{A}}(\alpha) = \text{LimSupAvg}(w(\alpha))$. In the sequel we also refer to the quantitative language $\mathcal{A}$ as the $\text{LimInfAvg}$ automaton $A$, and analogously the $\text{LimSupAvg}$ automaton $A$ is the quantitative language $\overline{\mathcal{A}}$.

**Mean-payoff automaton expressions.** A mean-payoff automaton expression $E$ is obtained by the following grammar rule:

$$E ::= \mathcal{A}[\overline{\mathcal{A}}] \max(E, E) \mid \min(E, E) \mid \text{sum}(E, E)$$

where $A$ is a deterministic (one-dimensional) weighted automaton. The quantitative language $L_E$ of a mean-payoff automaton expression $E$ is $L_E = \mathcal{A}$ (resp., $L_E = \overline{\mathcal{A}}$) if $E = \mathcal{A}$ (resp., if $E = \overline{\mathcal{A}}$), and $L_E = \text{op}(L_{E_1}, L_{E_2})$ if $E = \text{op}(E_1, E_2)$ for $\text{op} \in \{\max, \min, \text{sum}\}$. We shall, by convenient abuse of notation, interchangeably use $E$ to denote both the expression and the quantitative language of the expression (that is, $E$ will also denote $L_E$). An expression $E$ is called an atomic expression if $E = \mathcal{A}$ or $E = \overline{\mathcal{A}}$, where $A$ is a weighted automaton.

It was established in [31] (and it follows almost immediately by the construction of the class) that the class of mean-payoff automaton expressions is closed under max, min, sum and numerical complement.

**Decision problems and distance.** We consider the following classical decision problems for a quantitative language defined by a mean-payoff expression. Given a quantitative language $L$ and a threshold $\nu \in \mathbb{Q}$, the quantitative emptiness problem asks whether there exists a word $\alpha \in \Sigma^\omega$ such that $L(\alpha) \geq \nu$, and the quantitative universality problem asks whether $L(\alpha) \geq \nu$ for all words $\alpha \in \Sigma^\omega$.

Given two quantitative languages $L_1$ and $L_2$, the quantitative language-inclusion problem asks whether $L_1(\alpha) \leq L_2(\alpha)$ for all words $\alpha \in \Sigma^\omega$, and the quantitative language-equivalence problem asks whether $L_1(\alpha) = L_2(\alpha)$ for all words $\alpha \in \Sigma^\omega$. Finally, the distance between $L_1$ and $L_2$ is $D_{\text{sup}}(L_1, L_2) = \sup_{\alpha \in \Sigma^\omega} |L_1(\alpha) - L_2(\alpha)|$; and the corresponding computation problem is to compute the value of the distance.
Maximum value of expression. Given an expression $E$, its supremum value is the real number $\sup_{\alpha \in \Sigma^\omega} E(\alpha)$. While it is obvious that such supremum exists, it was proved in [31] that a maximum value also exists (that is, there exists $\alpha' \in \Sigma^\omega$ s.t. $E(\alpha') = \sup_{\alpha \in \Sigma^\omega} E(\alpha)$). Hence the maximum value of the expression $E$ is $\sup_{\alpha \in \Sigma^\omega} E(\alpha)$ or equivalently $\max_{\alpha \in \Sigma^\omega} E(\alpha)$.

Encoding of expressions and numbers. An expression $E$ is encoded by the tuple $(\langle E \rangle, \langle A_1 \rangle, \ldots, \langle A_k \rangle)$, where $\langle E \rangle$ is the expression string and $A_1, \ldots, A_k$ are the weighted automata that occur in the expression, w.l.o.g we assume that each automaton occur only once. A rational number is encoded as a pair of integers, where every integer is encoded in binary.

4.2 PSPACE Algorithm for Computing the Maximum Value of max-free Expressions

PSPACE Algorithm for Maximum Value In this section we consider only max-free expressions, which are expressions that contain only the min and sum operators. We will present a PSPACE algorithm that computes the maximum value of such expressions; computing the maximum value amounts to computing the maximum threshold for which the expression is nonempty; for this purpose we present four intermediate problems (and solutions), each problem is presented in a corresponding subsection below. The first problem asks whether an intersection of cut-point languages of $\text{LimSupAvg}$ automata is empty; the second problem asks the same question for $\text{LimInfAvg}$ automata; the third problem asks if an arbitrary intersection of cut-point languages of $\text{LimSupAvg}$ and $\text{LimInfAvg}$ automata is empty; and the last problem asks whether a max-free expression is empty. We will first present a naive solution for these problems; the solution basically lists all the simple cycles in the product automaton of the automata that occur in the expression; it then constructs linear constraints, with coefficients that depend on the weight vectors of the simple cycles, which their feasibility corresponds to the non-emptiness of the expression.

In the fifth subsection we will analyze the solution for the max-free emptiness problem; we will show a PSPACE algorithm that solves the problem; and we will bound the number of bits that are needed to encode the maximum threshold for which the expression is nonempty (recall that such maximal threshold is the maximum value of the expression); this will yield a PSPACE algorithm for computing the maximum value of a max-free expression.

In subsections 4.2.1-4.2.4 we shall assume w.l.o.g that the product automaton of all the automata that occur in the expression is a strongly connected graph; this can be done since in these subsections we do not refer to the complexity of the presented procedures.

4.2.1 The emptiness problem for intersection of $\text{LimSupAvg}$ automata

In this subsection we consider the problem where $k$ weighted automata $A_1, \ldots, A_k$ and a rational threshold vector $\vec{r} = (r_1, \ldots, r_k)$ are given, and we need to decide whether there
exists an infinite word $\alpha \in \Sigma^\omega$ such that $\overline{A}_i(\alpha) \geq r_i$ for all $i \in \{1, \ldots, k\}$; equivalently, whether the intersection $\bigcap_{i=1}^k \overline{A}_i \geq r_i$ is nonempty.

Informally, we prove that there is such $\alpha$ iff for every $i \in \{1, \ldots, k\}$, there is an infinite word $\alpha_i$ such that $\overline{A}_i(\alpha_i) \geq r_i$.

Formally, let $\mathcal{A} = A_1 \times \cdots \times A_k$ be the product automaton of the automata $A_1, \ldots, A_k$. Recall that an infinite word corresponds to an infinite path in $\mathcal{A}$, and that w.l.o.g we assume that the graph of $\mathcal{A}$ is strongly connected. Let $C_1, C_2, \ldots, C_n$ be the simple cycles that occur in $\mathcal{A}$. The next lemma claims that it is enough to find one cycle with average weight $r_i$ for every dimension $i$.

**Lemma 8.** There exists an infinite path $\pi$ in $\mathcal{A}$ such that $\overline{A}_i(\pi) \geq r_i$, for all $i \in \{1, \ldots, k\}$, iff for every $i \in \{1, \ldots, k\}$ there exists a simple cycle $C_i$ in $\mathcal{A}$, with average weight at least $r_i$.

**Proof.** The direction from left to right is easy:

Since for every $i \in \{1, \ldots, k\}$ there exists a path $\pi_i$ (namely $\pi_i$) such that $\overline{A}_i(\pi_i) \geq r_i$, it follows that there exists a simple cycle in $\mathcal{A}$ with average weight at least $r_i$ in dimension $i$. (This fact is well-known for one-dimensional weighted automata (e.g., see [110]), and hence it is true for the projection of $\mathcal{A}$ to the $i$-th dimension.)

For the converse direction, we assume that for every $i \in \{1, \ldots, k\}$ there exists a simple cycle $C_i$ in $\mathcal{A}$ with average weight at least $r_i$. Informally, we form the path $\rho$ by following the edges of the cycle $C_i$ until the average weight in dimension $i$ is sufficiently close to $r_i$, and then we do likewise for dimension $1 + (i \text{ mod } k)$, and so on. Formally, for every $\epsilon > 0$, and an arbitrary finite path $\lambda$ in $\mathcal{A}$, we construct the path $\pi^\epsilon(\lambda)$ in the following way: The first part of the path is $\lambda$, then we continue to a vertex in the cycle $C_1$ and follow the edges of the cycle $C_1$ until the average weight in the first dimension of the path is at least $r_1 - \epsilon$; we then continue to a vertex in $C_2$ and follow the edges of $C_2$ until the average weight in the second dimension is at least $r_2 - \epsilon$; we repeat the process also for $C_3, \ldots, C_k$. We recall that $\mathcal{A}$ is strongly connected, and therefore $\pi^\epsilon(\lambda)$ is a valid path. Note that for every $i \in \{1, \ldots, k\}$ there is a prefix of $\pi^\epsilon(\lambda)$ with average weight at least $r_i - \epsilon$ in dimension $i$. The reader can verify that the infinite path $\pi = \pi^{\frac{1}{k^2}} \pi^{\frac{1}{k}} \pi^{\frac{1}{k}} \cdots$ satisfies $\overline{A}_i(\pi) \geq r_i$ for every $i \in \{1, \ldots, k\}$, which concludes the proof of the lemma.

Lemma 8 shows that the emptiness problem for intersection of LimSupAvg automata can be naively solved by an exponential time algorithm that constructs the product automaton and checks if the desired cycles exist.

### 4.2.2 The emptiness problem for intersection of LimInfAvg automata

In this subsection we consider the problem where $k$ weighted automata $A_1, \ldots, A_k$ and a rational threshold vector $\vec{r} = (r_1, \ldots, r_k)$ are given, and we need to decide whether there exists an infinite word $\alpha \in \Sigma^\omega$ such that $\overline{A}_i(\alpha) \geq r_i$ for all $i \in \{1, \ldots, k\}$; or equivalently, whether the intersection $\bigcap_{i=1}^k \overline{A}_i \geq r_i$ is nonempty.
For $\text{LimInfAvg}$ automata, the componentwise technique we presented in the previous subsection will not work; to solve the emptiness problem for the intersection of such automata we need the notion of \( \vec{r} \) multi-cycles.

\section*{\( \vec{r} \) multi-cycles.}

Let \( G \) be a directed graph equipped with a multidimensional weight function \( w : E \rightarrow \mathbb{Q}^k \), and let \( \vec{r} \) be a vector of rationals. A \textit{multi-cycle} is a multi-set of simple cycles; the length of a multi-cycle \( C = \{C_1, \ldots, C_n\} \), denoted by \( |C| \), is \( \sum_{i=1}^{n} |C_i| \).

A multi-cycle \( C = \{C_1, \ldots, C_n\} \) is said to be an \( \vec{r} \) \textit{multi-cycle} if \( \frac{1}{|C|} \sum_{j=1}^{n} w(C_j) \geq \vec{r} \), that is, if the average weight of the multi-cycle, in every dimension \( i \), is at least \( r_i \).

In the sequel we will establish a connection between the problem of finding an \( \vec{r} \) multi-cycle and the emptiness problem for intersection of $\text{LimInfAvg}$ automata.

A polynomial time algorithm that decides if an \( \vec{r} \) multi-cycle exists is known \cite{64}; in this work however, it is sufficient to present the naive way for finding such multi-cycles; for this purpose we construct the following set of linear constraints:

Let \( C \) denote the set of all simple cycles in \( A \); for every \( c \in C \) we define a variable \( X_c \);

- we define the \( \vec{r} \) \textit{multi-cycle constraints} to be:

\[
\sum_{c \in C} X_c w(c) \geq \vec{r} ; \sum_{c \in C} |c| X_c = 1 ; \text{ and for every } c \in C : X_c \geq 0
\]

In the next lemma we establish the connection between the feasibility of the \( \vec{r} \) multi-cycle constraints and the existence of an \( \vec{r} \) multi-cycle.

\begin{lemma}
The automaton \( A \) has an \( \vec{r} \) multi-cycle iff the corresponding \( \vec{r} \) multi-cycle constraints are feasible.
\end{lemma}

\begin{proof}
The direction from left to right is immediate, indeed if we define \( X_c \) as the number of occurrences of cycle \( c \) in the witness \( \vec{r} \) multi-cycle divided by the length of that multi-cycle, then we get a solution for the set of constraints.

In order to prove the converse direction, it is enough to notice that if the constraints are feasible then they have a rational solution. Let \( \vec{X} \) be such rational solution, and let \( N \) be the least common multiple of all the denominators of the elements of \( \vec{X} \); by definition, the multi-set that contains \( NX_c \) copies of the cycle \( c \) is an \( \vec{r} \) multi-cycle.

\end{proof}

In the following two lemmas we establish a connection between the problem of finding an \( \vec{r} \) multi-cycle and the emptiness problem for intersection of $\text{LimInfAvg}$ automata. In the first lemma we show that if an \( \vec{r} \) multi-cycle does not exist, then for every infinite path there is a dimension \( i \) such that \( A_i(\pi) < r_i \).

\begin{lemma}
Let \( G = (V,E) \) be a directed graph equipped with a weight function \( w : E \rightarrow \mathbb{Q}^k \), and let \( \vec{r} \in \mathbb{Q}^k \) be a threshold vector. If \( G \) does not have an \( \vec{r} \) multi-cycle, then there exist constants \( \epsilon_G > 0 \) and \( m_G \in \mathbb{N} \) such that for every finite path \( \pi \) there is a dimension \( i \) for which \( w_i(\pi) \leq m_G + (r_i - \epsilon_G)|\pi| \).
\end{lemma}

\begin{proof}
Let \( \epsilon_G \) be the minimal \( \epsilon \) for which the \( \vec{r} - \epsilon \) multi-cycle constraints are feasible (where \( \vec{r} - \epsilon = (r_1 - \epsilon, \ldots, r_k - \epsilon) \)); note that \( \epsilon_G \) is the optimal solution for a linear
programming problem; the constraints of the linear programming problem are feasible, since the $\tilde{r} - \epsilon$ multi-cycle constraints are feasible for $\epsilon = 2W + \max\{r_1, \ldots, r_k\}$ (where $W$ is the maximal weight that occur in the graph); in addition the solution is bounded by $\epsilon = -W$; hence such $\epsilon_G$ must exist, and if $G$ does not have an $\tilde{r}$ multi-cycle then $\epsilon_G > 0$ (due to Lemma 9).

Let $\pi$ be an arbitrary finite path in $G$ of length longer than $|V|$, we decompose $\pi$ into three paths namely $\pi_0, \pi_e$ and $\pi_1$ such that $|\pi_0|, |\pi_1| \leq |V|$ and $\pi_e$ is a cyclic path (this can be done since any path longer then $|V|$ contains a cycle). Let $C_1, \ldots, C_n$ be the simple cycles that occur in $\pi_e$, and let $m_i$ be the number of occurrences of cycle $C_i$ in $\pi_e$. By definition we get that $\frac{1}{|\pi_0|}w(\pi_e) = \frac{1}{|\pi_0|} \sum_{j=1}^n m_j w(C_j)$; and $|\pi_e| = \sum_{j=1}^n m_j |C_j|$. Towards contradiction let us assume that there exists $\delta < \epsilon_G$ such that for every dimension $\frac{1}{|\pi_0|} w_i(\pi_e) \geq r_i - \delta$. Hence, by definition the $\tilde{r} - \delta$ multi-cycle constraints are feasible, which contradicts the minimality of $\epsilon_G$.

Therefore, for $m_G = -2|V|W$ (where $-W$ is the minimal weight that occur in the graph) we get that for every finite path $\pi$ there exists a dimension $i$ for which $w_i(\pi) \leq m_G + (r_i - \epsilon_G)|\pi|$.

\[\square\]

**Lemma 11.** There exists an infinite path $\pi$ in $A$ such that $A_i(\pi) \geq r_i$, for all $i \in \{1, \ldots, k\}$, iff the graph of $A$ contains an $\tilde{r} = (r_1, \ldots, r_k)$ multi-cycle.

**Proof.** To prove the direction from right to left we use Lemma 10. Lemma 10 implies that if an $\tilde{r}$ multi-cycle does not exists, then for every infinite path $\pi$ there exist a dimension $i$ and an infinite sequence of indices $j_1 < j_2 < j_3 \ldots$ such that the average weight of the prefix of $\pi$, of length $j_m$, is at most $r_i - \frac{\epsilon_G}{2}$, for all $m \in \mathbb{N}$. Hence by definition $\text{LimInfAvg}_i(\pi) < r_i$.

In order to prove the converse direction, let us assume that $G$ has an $\tilde{r}$ multi-cycle $C = C_1, \ldots, C_n$, such that the cycle $C_i$ occurs $m_i$ times in $C$. We obtain the witness path $\pi$ in the following way (we demonstrate the claim for $n = 2$): let $\pi_{12}$ be a path from $C_1$ to $C_2$ and $\pi_{21}$ be a path from $C_2$ to $C_1$ (recall that the graph is strongly connected), we define

$$\pi = C_1^{m_1} \pi_{12} C_2^{m_2} \pi_{21} (C_1^{m_1})^2 \pi_{12} (C_2^{m_2})^2 \pi_{21} \ldots$$

Informally, the long-run average weight of the path $\pi$ is determined only by the cycles $C_1^{m_1}$ and $C_2^{m_2}$, since the effect of the paths $\pi_{12}$ and $\pi_{21}$ on the average weight of a prefix of $\pi$ becomes negligible as the length of the prefix tends to infinity. Thus $A_i(\pi) \geq r_i$ for every dimension $i$, which concludes the proof of Lemma 11.

\[\square\]

Lemma 11 and Lemma 9 immediately give us the following naive algorithm for the emptiness problem for intersection of LimInfAvg automata: First, construct the product automaton; second, list all the simple cycles in the product automaton; third, construct the $\tilde{r}$ multi-cycle constraints and check for their feasibility.
When the automata and the threshold vector are clear from the context, we shall refer to the $\vec{r}$ multi-cycle constraints, which are constructed from the intersection of the given Limit Avg automata and the threshold $\vec{r}$, as the \textit{lim-inf constraints}.

4.2.3 The emptiness problem for intersection of \textit{LimSupAvg} and \textit{LimInfAvg} automata

In this subsection we consider the problem where $2k$ weighted automata $A_1, A_2, \ldots, A_k, B_1, B_2, \ldots, B_k$ and two $k$-dimensional rational threshold vectors $\vec{r}^a$ and $\vec{r}^b$ are given, and we need to decide whether there exists an infinite word $\alpha \in \Sigma^\omega$ such that $A_i(\alpha) \geq r_i^a$ and $B_i(\alpha) \geq r_i^b$ for all $i \in \{1, \ldots, k\}$; or equivalently, whether the intersection $(\bigcap_{i=1}^k A_i \geq r_i^a) \cap (\bigcap_{i=1}^k B_i \geq r_i^b)$ is nonempty.

Our solution will be a result of the following two lemmata. The first lemma claims that there is a word that satisfies all the lim-inf conditions if there are words $\alpha_1, \ldots, \alpha_k$ such that $\alpha_j$ satisfies all the lim-inf conditions and the lim-sup condition for the automaton $B_j$.

\textbf{Lemma 12.} There exists an infinite word $\alpha$ for which $A_i(\alpha) \geq r_i^a$ and $B_i(\alpha) \geq r_i^b$ for all $i \in \{1, \ldots, k\}$ iff there exist $k$ infinite words $\alpha_1, \alpha_2, \ldots, \alpha_k$ such that for every $j \in \{1, \ldots, k\}$:

$$A_i(\alpha_j) \geq r_i^a \text{ for all } i \in \{1, \ldots, k\}; \text{ and } B_j(\alpha_j) \geq r_j^b$$

\textbf{Proof.} In this proof, w.l.o.g, we assume that both threshold vectors are the zero vector (that is, the vector $\vec{0}$).

The proof for the direction from left to right is trivial. To prove the converse direction, we denote by $A$ the $2k$-dimensional product automaton of the automata $A_1, \ldots, A_k, B_1, \ldots, B_k$; recall that an infinite word corresponds to an infinite path in $A$; hence we can assume that there exist infinite paths $\pi_1, \ldots, \pi_k$ such that for every $j \in \{1, \ldots, k\}$:

$$A_i(\pi_j) \geq 0 \text{ for all } i \in \{1, \ldots, k\}; \text{ and } B_j(\pi_j) \geq 0$$

Informally, we shall construct the witness path $\pi$, for which $A_i(\pi) \geq 0$ and $B_i(\pi) \geq 0$ for all $i \in \{1, \ldots, k\}$, by following the path $\pi_j$ until the average weights in the corresponding dimensions of $A_1, \ldots, A_k$ and $B_j$ are sufficiently close to $0$, and repeat the process for $1 + j \mod k$, and so on.

Formally: We denote the value of the minimal weight that occur in the graph by $-W$. For every $j \in \{1, \ldots, k\}$ and $\epsilon > 0$ we denote by $N_j^\epsilon$ the first position in the path $\pi_j$ such that in every position of $\pi_j$, that is greater than $N_j^\epsilon$, the average weight in all the dimensions that corresponds to $A_1, \ldots, A_k$ is at least $-\epsilon$; the reader should note that by definition, such $N_j^\epsilon$ always exists; we denote $N_j^\epsilon = \max_{j \in \{1, \ldots, k\}} N_j$.

For every $m \in \mathbb{N}$, $\epsilon > 0$ and a finite path $\lambda$ we denote by $\pi_j^{m,\epsilon,\lambda}$ the shortest prefix of the path $\lambda \cdot \pi_j$, of length at least $m$, such that average weights in the prefix, in all the dimensions that correspond to $A_1, \ldots, A_k$ and $B_j$, are at least $-\epsilon$ (note that by the
definition of $\pi_j$, such prefix exists). The following remark demonstrate the key property of the definitions above.

**Remark 3.** Let $\lambda$ be a finite path, and let $M_\epsilon = \frac{(W,N)^{\epsilon}}{\epsilon}$. For every $i, j \in \{1, \ldots, k\}$ and $\epsilon > 0$, the infinite path $\pi_i^{M_\epsilon,\epsilon,\lambda}$ satisfies

In every position of the path, greater than $\pi_i^{M_\epsilon,\epsilon,\lambda}$, the average weight in the dimensions that correspond to $A_1, \ldots, A_k$ is at least $-2\epsilon$.

We define an infinite sequence of finite paths $\lambda_0, \lambda_1, \lambda_2, \ldots$, in the following way:

- $\lambda_0$ is an arbitrary path, for example: the first edge of $\pi_1$.
- For $i > 0$: we denote $\lambda_i = \lambda_{i-1} \cdot \pi_i^{M_\epsilon_i,\epsilon_i,\lambda_{i-1}}$, where $\epsilon_i = \frac{1}{i}$ and for $i > k$:
  $$\pi_i \equiv \pi_{1+(i \mod k)}.$$  

We define $\pi$ to be the limit path of the sequence $\{\lambda_i\}_{i=1}^\infty$: by the construction of $\pi$ it follows that for every $\epsilon > 0$, in infinite many positions, the average weight of the dimension that corresponds to $B_i$ is at least $-\epsilon$, and that as of certain position, the average weight in all the dimensions that correspond to $A_1, \ldots, A_k$ is at least $-\epsilon$. Hence, by definition, for all $i \in \{1, \ldots, k\}$:

$$A_i(\pi) \geq 0 \text{ and } B_i(\pi) \geq 0$$

as required, and the proof of the lemma follows.

The second lemma shows that the emptiness problem for an intersection of lim-inf automata and one lim-sup automaton can be reduced to the emptiness problem for an intersection of lim-inf automata.

**Lemma 13.** The intersection $\overline{B_1^{r_1^1} \cap (\bigcap_{i=1}^k A_i^{r_i^1})}$ is nonempty iff the intersection $\overline{B_1^{r_1^1} \cap (\bigcap_{i=1}^k A_i^{r_i^1})}$ is nonempty.

**Proof.** The direction from right to left is trivial: to prove the converse direction, let us assume that the intersection $\overline{B_1^{r_1^1} \cap (\bigcap_{i=1}^k A_i^{r_i^1})}$ is empty, and we shall prove that the intersection $\overline{B_1^{r_1^1} \cap (\bigcap_{i=1}^k A_i^{r_i^1})}$ is also empty. W.l.o.g we assume that $r_1^1 = r_0^1 = 0$. Let $\mathcal{A}$ be the $k+1$-dimensional product automaton of the automata $A_1, \ldots, A_k$ and $B_1$. Let $\pi$ be an infinite path in $\mathcal{A}$, we will show that either $A_i(\pi) < 0$ for some $i \in \{1, \ldots, k\}$ or that $\overline{B_1(\pi)} < 0$. We first observe that the automaton $\mathcal{A}$ does not have a $\bar{0}$ multi-cycle; this fact follows from the assumption that the intersection $\overline{B_1^{r_1^1} \cap (\bigcap_{i=1}^k A_i^{r_i^1})}$ is empty and from Lemma 11. Hence, by Lemma 10, there exist constants $m \in \mathbb{N}$ and $c > 0$ such that for every prefix $\pi'$ of the path $\pi$ there exists a dimension $i$ for which the weight of $\pi'$ in dimension $i$ is at most $m - c|\pi'|$; let $j$ be the corresponding dimension of the automaton $B_1$, and let $\pi^*$ be the longest prefix with average weight at most $-\frac{c}{2}$ in dimension $j$ (note that $\pi^*$ does not necessarily exist). We consider two disjoint cases: In the first case, there exists such $\pi^*$, and then, by definition $\overline{B_1(\pi)} \leq -\frac{c}{2} < 0$.
and the claim follows. In the second case, the path $\pi$ has infinitely many prefixes with average weight at least $-\frac{c}{2}$ in dimension $j$; therefore (by Lemma 10) there must exist a dimension $i$ for which the average weight in dimension $i$ is at most $-\frac{c}{2}$ for infinitely many prefixes of $\pi$; hence, $A_i(\pi) \leq -\frac{c}{2} < 0$, and the claim follows.

To conclude, if the intersection $B_1 \geq 0 \cap (\bigcap_{i=1}^{k} A_i \geq 0)$ is empty, then for every infinite path $\pi$, either $B_1(\pi) < 0$ or there exists some $i \in \{1, \ldots, k\}$ such that $A_i(\pi) < 0$, which concludes the proof of Lemma 13.

Due to Lemma 12 and 13 we can solve the emptiness problem for intersection of lim-inf and lim-sup automata in the following way: First, we construct the product automata $A_i = A_1 \times \cdots \times A_k \times B_i$ for all $i \in \{1, \ldots, k\}$ and list all the simple cycles that occur in it; second, we construct the threshold vector $\vec{r} = (r_1^a, \ldots, r_k^a, r_i^b)$ and check if the graph of $A^i$ has an $\vec{r}^i$ multi-cycle, for all $i \in \{1, \ldots, k\}$. Due to Lemma 12 and 13 the intersection is nonempty iff every $A^i$ has an $\vec{r}^i$ multi-cycle, that is, if the $\vec{r}^i$ multi-cycle constraints are feasible.

Recall that the existence of an $\vec{r}^i$ multi-cycle in the graph of $A^i$ is equivalent to the feasibility of the corresponding lim-inf constraints for $A^i$ and $\vec{r}^i$; in the sequel, we will refer to the set of constraints $\bigcup_{i=1}^{k}\{\text{lim-inf constraints for } A^i \text{ and } \vec{r}^i\}$ as the min-only constraints. (As we use them to decide the emptiness of expressions that contain only the min operator.)

In this subsection we proved that the emptiness of the intersection of LimInfAvg and LimSupAvg automata is equivalent to the feasibility of the corresponding min-only constraints.

4.2.4 The emptiness problem for max-free expressions

In this subsection we solve the emptiness problem for max-free expressions. The solution we present is a reduction to the emptiness problem for an intersection of lim-inf and lim-sup automata with a threshold vector that satisfies certain linear constraints; the reduction yields a naive double-exponential complexity upper-bound for the problem, which we will improve in the succeeding subsection.

The reduction is based on the next simple observation.

**Observation 1.** The expression $E = E_1 + E_2$ is nonempty with respect to the rational threshold $\nu$ iff there exist two thresholds $\nu_1, \nu_2 \in \mathbb{R}$ such that (i) The intersection of the cut-point languages $E_1^{\geq \nu_1}$ and $E_2^{\geq \nu_2}$ is nonempty; and (ii) $\nu_1 + \nu_2 \geq \nu$.

If $E = E_1 + E_2$ and $E_1$ and $E_2$ are min-only expressions then we decide the emptiness of $E$ in the following way: we combine the min-only constraints for the expressions $E_1$ and $E_2$ with respect to arbitrary thresholds $r_1$ and $r_2$ (that is, $r_1$ and $r_2$ are variables in the constraints), note that these are still linear constraints; we then check the feasibility of the constraints subject to $r_1 + r_2 \geq \nu$. (Note that as all the constraints are linear, this can be done by linear programming.)
The next lemma shows that in the general case, the emptiness problem for an arbitrary max-free expression and a threshold $\nu$ can be reduced, in polynomial time, to the emptiness problem for an intersection of lim-inf and lim-sup automata with respect to threshold vectors $r^a$ and $r^b$ subject to certain linear constraints on $r^a$ and $r^b$.

**Lemma 14.** Let $E$ be a max-free expression with atomic expressions $e_1, \ldots, e_k$, and let $\nu$ be a rational threshold, then there exist a $2k \times 2k$ matrix $M_E$ and a $2k$-dimensional vector $\vec{b}_\nu$, with rational coefficients, where $M_E$ and $\vec{b}_\nu$ are computable in polynomial time (from $E$ and $\nu$) such that:

The expression $E$ is nonempty (with respect to $\nu$) iff there exists a 2$k$-dimensional vector of reals $\vec{r}$ such that the intersection $\bigcap_{i=1}^k e_i^{\geq \nu_i}$ is nonempty and $M_E \times \vec{r} \geq \vec{b}_\nu$.

Instead of formally proving the correctness of Lemma 14, we provide a generic example that illustrates the construction of the matrix $M_E$ and the vector $\vec{b}_\nu$.

**Example 1.** Let $E = \min(A_1, (A_2 + A_3)) + \min(A_4, A_5)$. Then for every $\nu \in \mathbb{R}$, each the following condition is equivalent to $E^{\geq \nu} \neq \emptyset$.

- $\exists r_6, r_7 \in \mathbb{R}$ such that $L_{\min(A_1, (A_2 + A_3))}^{\geq r_6} \cap L_{\min(A_4, A_5)}^{\geq r_7} \neq \emptyset$ and $r_6 + r_7 \geq \nu$.
- $\exists r_1, r_4, r_5, r_6, r_7, r_8$ such that $A_1^{r_1} \cap L_{A_2 + A_3}^{\geq r_6} \cap \overline{A_4}^{\geq r_4} \cap \overline{A_5}^{\geq r_5} \neq \emptyset$ and $r_1 \geq r_6$, $r_8 \geq r_6$, $r_4 \geq r_7$, $r_5 \geq r_7$ and $r_6 + r_7 \geq \nu$.
- $\exists r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8$ such that $A_1^{r_1} \cap A_2^{r_2} \cap \overline{A_3}^{r_3} \cap \overline{A_4}^{r_4} \cap \overline{A_5}^{r_5} \neq \emptyset$ and $r_1 \geq r_6$, $r_2 + r_3 \geq r_8$, $r_9 \geq r_6$, $r_4 \geq r_7$, $r_5 \geq r_7$ and $r_6 + r_7 \geq \nu$.

The reader should note that we associate every variable $r_i$ either with a sub-expression or with an atomic expression; as we assume that each atomic expression occurs only once, the number of variables is at most $2k$.

Hence we can solve the emptiness problem for a rational threshold $\nu$ and a max-free expression $E$, which contains the atomic expressions $e_1, \ldots, e_k$ in the following way: First, we construct the matrix $M_E$ and the vector $\vec{b}_\nu$; second, we construct the min-only constraints for the intersection $\bigcap_{i=1}^k e_i^{\geq \nu_i}$ and check for their feasibility subject to the constraints $M_E \times \vec{r} \geq \vec{b}_\nu$.

In the sequel we will refer to the min-only constraints along with the $M_{E, \nu} \times \vec{r} \geq \vec{b}_\nu$ constraints as the **max-free constraints**; we will show that even though the size of the constraints is double-exponential, there is a PSPACE algorithm that decides their feasibility (when the input is $E$ and $\nu$).

### 4.2.5 PSPACE algorithm for the emptiness problem of max-free expressions

In this subsection we will present a PSPACE algorithm that for given max-free expression $E$ and rational threshold $\nu$, decides the feasibility of the max-free constraints;
as shown in subsection 4.2.4, such algorithm also solves the emptiness problem for max-free expressions. Informally, we will show that if the max-free constraints are feasible then they have a short solution, and that a short solution can be verified by a polynomial-space machine; hence the problem is in NPSPACE, and due to Savitch Theorem, also in PSPACE.

The next lemma describes key properties of max-free constraints, which we will use to obtain the PSPACE algorithm.

**Lemma 15.** For every max-free expression $E$:

1. For every threshold $\nu$, the max-free constraints have at most $O(k^2)$ constraints, where $k$ is the number of automata that occur in $E$, that are not of the form of $x \geq 0$, where $x$ is a variable.

2. There exists a bound $t$, polynomial in the size of the expression, such that for every threshold $\nu$, the max-free constraints are feasible iff there is a solution that assigns a nonzero value to at most $t$ variables.

3. There exists a bound $t$, polynomial in the size of the expression, such that the maximum threshold $\nu \in \mathbb{R}$, for which the max-free constraints are feasible, is a rational and can be encoded by at most $t$ bits. (In particular such maximum $\nu$ exists.)

**Proof.** To prove the first item of Lemma 15 we present a detailed description of the max-free constraints for an expression $E$ and a threshold $\nu$. Let $\mathcal{A}$ be the $k$-dimensional product automaton of the automata that occur in the expression $E$; w.l.o.g we assume that for some $j$, the first $j$ dimensions of $\mathcal{A}$ correspond to the LimInfAvg automata $A_1, \ldots, A_j$, and the last $k - j$ dimensions correspond to the LimSupAvg automata $A_{j+1}, \ldots, A_k$. Let $\vec{r}$ be a $2k$-dimensional vector of variables; let $\mathcal{C}$ be the set of all simple cycles in $\mathcal{A}$; and we define a variable $X_c^i$ for every cycle $c$ in $\mathcal{C}$ and a dimension $i \in \{j+1, \ldots, k\}$; then the min-only constraints for the expression $E$ and the threshold vector $(r_1, \ldots, r_k)$ are as follows:

$$\sum_{c \in \mathcal{C}} X_c^i w_m(c) \geq r_m \text{ for every } i \in \{j+1, \ldots, k\}, m \in \{1, \ldots, j, i\} \quad (4.1)$$

$$\sum_{c \in \mathcal{C}} |c| X_c^i = 1 \text{ for every } i \in \{j+1, \ldots, k\} \quad (4.2)$$

$$X_c^i \geq 0 \text{ for every } i \in \{j+1, \ldots, k\}, \text{ and } c \in \mathcal{C} \quad (4.3)$$

and the max-free constraints are the min-only constraints along with the constraints

$$M_E \times \vec{r} \geq \vec{b}_\nu \quad (4.4)$$

where $M_E$ and $\vec{b}_\nu$ are the $2k \times 2k$ matrix and the $2k$-dimensional vector from Lemma 14.
The proof of Lemma 15(1) follows immediately from the definition of the max-free constraints.

To prove the additional two items of the lemma, we observe that the max-free constraints for the expression $E$ and the threshold $\nu$ remain linear when the threshold $\nu$ is a variable; hence, the problem of computing the maximum threshold $\nu$ for which the max-free constraints are feasible amounts to the following linear-programming problem:

Find the maximum $\nu$ subject to the max-free constraints for expression $E$ and threshold $\nu$.

Let $W$ be a bound on the absolute value of the weights of $\mathcal{A}$; clearly the max-free constraints are feasible for the threshold $-W$, and are infeasible for the threshold $+2W$; hence, the domain of feasible thresholds is bounded and nonempty, and by standard properties of linear programming [75], it follows that a maximum threshold exists. Moreover, we observe that every coefficient in the max-free constraints can be encoded by polynomial number of bits (as the length of a simple cycle in the product automaton is at most exponential in the size of the input), and that the number of constraints, which are not of the form of $x \geq 0$, is $O(k^2)$; we denote by $D$ the maximal coefficient in the max-free constraints, by standard properties of linear programming [75], it follows that the maximum value of $\nu$ is obtained when at most $O(k^2)$ variables are assigned with rational nonzero values, and the result of the linear-programming (that is, the maximum threshold) is a rational with numerator and denominator bounded by $D^{O(k^2)}$; therefore only $O(k^2) \log(D)$ bits are required to encode the maximum threshold.

To conclude, we proved that for the maximum threshold $\nu^*$ for which the max-free constraints are feasible, there is a solution for the corresponding max-free constraints such that at most $O(k^2)$ variables are assigned with nonzero values; and the threshold $\nu^*$ requires only polynomial number of bits to encode; hence the last two items of Lemma 15 immediately follows.

Recall that a rational solution for the max-free constraints corresponds to vectors of thresholds and a set of multi-cycles, each multi-cycle with an average weight that matches its corresponding threshold vector; by Lemma 15(2) the number of different simple cycles that occur in the witness multi-cycles set is at most $t$. We also observe that if a multi-set of cycles (that are not necessarily simple) with average weight vector $\vec{\nu}$ exists, then a $\vec{\nu}$ multi-cycle (of simple cycles) also exists, since we can decompose every non-simple cycle to a set of simple cycles; thus, a $\vec{\nu}$ multi-cycle exists iff there exists a multi-set of short cycles, where the length of each cycle in the multi-set is at most the number of vertices in the graph (note that in particular, every simple cycle is short).

Hence, we can decide the feasibility of the max-free constraints in the following way: First, we guess $t$ weight vectors of $t$ short cycles that occur in the same strongly connected component (SCC) of the product automaton of all the automata that occur
in the expression; second, we construct the $O(k^2)$ constraints of the max-free constraints and assign zero values to all the variables of the non-chosen cycles; third, we check the feasibility of the formed $O(k^2)$ constraints, where each constraint has at most $t + 1$ variables.

Note that we can easily perform the last two steps in polynomial time (as the values of the weight vector of every short cycle can be encoded by polynomial number of bits); hence, to prove the existence of a PSPACE algorithm, it is enough to show how to encode (and verify by a polynomial-space machine) $t$ average weight vectors of $t$ short cycles that belong to one SCC of the product automaton. Informally, the encoding scheme is based on the facts that every vertex in the product automaton is a $k$-tuple of states, and that a path is a sequence of alphabet symbols; the verification is done by simulating the $k$ automata in parallel, and since the size of the witness string should be at most exponential, we can do it with a polynomial-size tape.

To conclude, we proved that there is a PSPACE algorithm that decides the feasibility of the max-free constraints, and therefore the next lemma follows.

**Lemma 16.** The emptiness problem for max-free expressions is in PSPACE.

**Proof.** Recall that we can decide the non-emptiness of a max-free expression by checking the feasibility of its corresponding max-free constraints, and that a rational solution for the max-free constraints corresponds to vectors of thresholds and a set of multi-cycles, each multi-cycle with an average weight that matches its corresponding threshold vector; by Lemma 15(2) the number of different simple cycles that occur in the witness multi-cycles set is at most $t$. We also observe that if a multi-set of cycles (that are not necessarily simple) with average weight vector $\vec{\nu}$ exists, then a $\vec{\nu}$ multi-cycle (of simple cycles) also exists, since we can decompose every non-simple cycles to a set of simple cycles; thus, a $\vec{\nu}$ multi-cycle exists iff there exists a multi-set of short cycles, where the length of each cycle in the multi-set is at most the number of vertices in the graph (note that in particular, every simple cycle is short).

Hence, we can decide the feasibility of the max-free constraints in the following way: First, we guess $t$ weight vectors of $t$ short cycles that occur in the same SCC of the product automaton of all the automata that occur in the expression; second, we construct the $O(k^2)$ constraints of the max-free constraints and assign zero values to all the variables of the non-chosen cycles; third, we check the feasibility of the formed $O(k^2)$ constraints, where each constraint has at most $t + 1$ variables.

Note that we can easily perform the last two steps in polynomial time (as the values in the weight vector of every short cycle can be encoded by polynomial number of bits); hence, to prove the existence of a PSPACE algorithm, it is enough to show how to encode (and verify by a polynomial-space machine) $t$ average weight vectors of $t$ short cycles that belong to one SCC of the product automaton.

The encoding technique is straight forward and standard; we present its details only for the purpose of self-containment. We based the encoding on the observations
that a vertex in the product automaton is a $k$-tuple of states, and that a short cycle is characterized by: (a) its length, which is at most the size of the product automaton (that is, at most exponential in the size of the expression); (b) its initial (and final) vertex (that is, a $k$-tuple of states); and (c) a sequence of alphabet symbols that corresponds to the path from its initial vertex to its end vertex.

Thus, we encode $t$ average weights by the sequence

$$(w_1, \ldots, w_t)\#(a_1, b_1, c_1)\#(a_2, b_2, c_2)\# \ldots \#(a_t, b_t, c_t)\#\pi_1\rightarrow 2\rightarrow \cdots \rightarrow t\rightarrow 1$$

where $w_1, \ldots, w_t$ are the $t$ average weights; $a_i, b_i$ and $c_i$ are respectively the length, initial vertex and the path (that is, sequence of alphabet symbols) of the cycle with the weight $w_i$; and $\pi_1\rightarrow 2\rightarrow \cdots \rightarrow t\rightarrow 1$ is a sequence of symbols that corresponds to a cyclic path between all the initial vertices of all $t$ cycles.

To verify the encoding, we simply simulate, in parallel, each of the $k$ weighted automata, and store the weight vectors of each short cycle, which requires at most polynomial number of bits; we verify that all the cycles are in the same SCC, given the witness $\pi_1\rightarrow 2\rightarrow \cdots \rightarrow t\rightarrow 1$, in similar way; finally, all that is left is to verify that the vector $(w_1, \ldots, w_t)$ corresponds to the real average weights of the cycles; in addition, we reject the witness if its number of bits exceeds the (at most exponential) length threshold of $t^3|A|$, where $A$ is the product automaton of all the automata occur in the expression.

To conclude, we proved that there is a PSPACE algorithm that decides the feasibility of the max-free constraints, and Lemma 16 follows.

Lemma 16 along with Lemma 15(3) imply a PSPACE algorithm that computes the maximum value of a max-free expression; the next lemma formally states this claim.

**Lemma 17.** (i) The maximum value of a max-free expression is a rational value that can be encoded by polynomial number of bits (in particular, every expression has a maximum value); and (ii) The maximum value of a max-free expression is PSPACE computable.

**Proof.** By definition, there exists an infinite word $\alpha$ such that $E(\alpha) \geq \nu$ iff the expression $E$ is nonempty with respect to threshold $\nu$. By Lemma 15(3) and by the equivalence of the feasibility of the max-free constraints and the emptiness of the expression, the maximum threshold for which the expression $E$ is nonempty is a $t$-bits rational number (and moreover such maximum threshold exists); hence we can find this maximum threshold by checking the emptiness of $E$ for all thresholds that are $t$-bits rational numbers, and return the maximal such threshold (possible optimization is to

---

\footnote{Note that in general, the sum of exponential number of rationals may require exponential number of bits; however, in our case, there are only $|A|$ different rational numbers, hence their lowest common denominator has only polynomial number of bits, and their sum can be encoded by polynomial number of bits.}
do binary search over all \( t \)-bits thresholds); this can be done in polynomial space due to Lemma 16.

\[ \square \]

\section{The Complexity of Mean-Payoff Expression Problems}

\subsection*{Complexity}

In this section we will prove PSPACE membership, and PSPACE hardness, for the classical mean-payoff expression problems; the key step in the proof of the PSPACE membership is the next theorem, which extends Lemma 17 to arbitrary expressions (as opposed to only max-free expressions).

\begin{theorem}
\( (i) \) The maximum value of an expression is a rational value that can be encoded by polynomial number of bits (in particular, every expression has a maximum value); and \( (ii) \) The maximum value of an expression is PSPACE computable.
\end{theorem}

\begin{proof}
Informally, we prove that if the number of max operators in the expression \( E \) is \( m > 0 \), then we can construct in linear time two expressions \( E_1 \) and \( E_2 \), each with at most \( m - 1 \) max operators and of size at most \( |E| \), such that \( E = \max(E_1, E_2) \); hence, in order to compute the maximum value of \( E \), we recursively compute the maximum values of \( E_1 \) and \( E_2 \), and return the maximum of the two values; note that if the expression is max-free (that is, if \( m = 0 \)), then thanks to Lemma 17, the maximum value is PSPACE computable and can be encoded by polynomial number of bits.

We now give the formal proof. First, we claim that if the number of max operators in the expression \( E \) is \( m > 0 \), then we can construct in linear time two expressions \( E_1 \) and \( E_2 \), each with at most \( m - 1 \) max operators and of size at most \( |E| \), such that \( E = \max(E_1, E_2) \). We prove the claim by induction on the number of (any) operators in the expression: if the number of operators in the expression is zero, the claim is trivially satisfied; otherwise let \( E = \text{op}(F, G) \) such that the expression \( F \) has at least one max operator; as \( F \) has strictly fewer operators (as compared to \( E \)), by the inductive hypothesis there exist two expressions \( F_1 \) and \( F_2 \), each with at most \( m - 1 \) max operators, and of length at most \( |F| \), such that \( F = \max(F_1, F_2) \); the reader can verify that \( E_1 = \text{op}(F_1, G) \) and \( E_2 = \text{op}(F_2, G) \) satisfy the claim, that is, \( E = \max(E_1, E_2) \) and \( |E_1|, |E_2| \leq |E| \).

Second, we observe the fact that if \( E = \max(E_1, E_2) \), then

\[ \sup_{\alpha \in \Sigma^\omega} E(\alpha) = \max(\sup_{\alpha \in \Sigma^\omega} E_1(\alpha), \sup_{\alpha \in \Sigma^\omega} E_2(\alpha)) \]

Hence, in order to compute the maximum value of \( E \), we simply compute the maximum value of \( E_1 \), the maximum value of \( E_2 \) and return the maximum of the two values. If \( E_1 \) and \( E_2 \) are max-free expressions, then thanks to Lemma 17 we can compute their maximum values by polynomial space Turing machine; in the general case, if the number of max operators in the expression \( E \) is \( m \), then we construct in linear time two expressions \( E_1 \) and \( E_2 \), each with at most \( m - 1 \) max operators, such that \( E = \max(E_1, E_2) \); then we recursively compute the maximum value of \( E_1 \) and the maximum value of \( E_2 \) and return the maximum of these values.

The reader can easily verify that the procedure we described requires only polynomial space to run, and thus the proof of Theorem 3 follows.

The PSPACE membership of the classical problems (namely, emptiness, universality, inclusion, equivalence and distance) follows almost trivially from Theorem 3. Indeed, for a given threshold, an expression is empty if its maximum value is less than the threshold, and an expression is universal if its minimum value (that is, the maximum of its numerical complement) is not less than the threshold; the language inclusion and equivalence problems are special cases of the universality problem (since the class of mean-payoff expressions is closed under numerical complement and the sum operator); and the distance of the expressions $E_1$ and $E_2$ is the maximum value of the expression $F = \max(E_1 - E_2, E_2 - E_1)$.

The PSPACE lower bounds for the decision problems are obtained by reductions from the emptiness problem for intersection of regular languages (see proofs in the next section), which is PSPACE-hard [67].

Thus, we get the main result of this chapter:

**Theorem 4.** For the class of mean-payoff automaton expressions, the quantitative emptiness, universality, language inclusion, and equivalence problems are PSPACE-complete, and the distance is PSPACE computable.

### 4.4 Lower bounds

In this section we will establish PSPACE lower bounds for the emptiness, universality, language inclusion and language equivalence problems; we obtain the PSPACE lower bound for the emptiness problem by a reduction from the emptiness problem for intersection of regular languages (which is PSPACE-hard [67]); for the universality problem we obtain the bound by a reduction from the universality problem for union of regular languages (which is the complement of the first problem, and hence, also PSPACE-hard); and the lower bounds for the language inclusion and equivalence problems are obtained by a reduction from the universality problem (of mean-payoff expressions).

To present the reduction from the emptiness problem for intersection of regular languages we define, for a language of finite words $L \subseteq \Sigma^*$, and for a symbol $\xi \notin \Sigma$, the following function, for every infinite word $\alpha \in (\Sigma \cup \{\xi\})^\omega$:

$$f^L(\alpha) = \begin{cases} +1 & \text{if } \alpha \in L \cdot \xi \cdot (\Sigma + \xi)^\omega \\ -1 & \text{otherwise} \end{cases}$$

It is easy to verify that if the language $L$ is recognizable by a finite-state automaton $A$, then we can construct in linear time a weighted automaton $A$ such that $A \equiv f^L(A)$; and that for any $k$ finite-state automata $A_1, \ldots, A_k$ over the alphabet $\Sigma$, the intersection $\bigcap_{i=1}^k L(A_i)$ is nonempty iff the expression $E = \min(A_1, \ldots, A_k)$ is nonempty with
respect to threshold 0 and alphabet \( \Sigma \cup \{\xi\} \); thus, the emptiness problem for mean-payoff expressions is PSPACE-hard.

We prove the PSPACE hardness of the universality problem in a similar way; we define the next function for every infinite word \( \alpha \in (\Sigma \cup \{\xi\})^\omega \):

\[
g(\alpha) = \begin{cases} +1 & \text{if } \alpha \in \Sigma^\omega \\ -1 & \text{otherwise} \end{cases}
\]

we denote by \( G \) the (minimal) weighted automaton for which \( G(\alpha) \equiv g \) (surely, such automaton exists); and we observe that the union of the regular languages \( \bigcup_{i=1}^k L(A_i) \) is universal iff the expression \( E = \max(A_1, \ldots, A_k, G) \) is universal with respect to threshold 0 and alphabet \( \Sigma \cup \{\xi\} \) (recall that \( A_i \equiv f_{L(A_i)} \)). Thus, the universality problem for mean-payoff expressions is PSPACE-hard.

The reductions from the universality problem to the language inclusion and equivalence problems are trivial; let \( Z \) denote a weighted automaton for which \( Z(\alpha) \equiv 0 \) (for example an automaton where all the weights of the edges are zero, and let \( E_0 \) denote \( Z \); then the expression \( E \) is universal with respect to threshold 0 iff \( E \geq E_0 \), and iff the expression \( \min(E, E_0) \) is equivalent to the expression \( E_0 \). Since the universality problem is PSPACE-hard even for threshold 0, the next lemma follows.

**Lemma 18.** For the class of mean-payoff automaton expressions, the quantitative emptiness, universality, language inclusion, and equivalence problems are PSPACE-hard.
Chapter 5

Finite-Memory Strategy Synthesis for Multidimensional Mean-Payoff Objectives

In the quantitative setting, there are two relevant synthesis problems: (i) the \textit{quantitative analysis problem} is to compute the optimal (infimum) value that a player-1 strategy can assure; and (ii) the \textit{boolean analysis decision problem} is to determine whether player 1 can assure a value of at most $\nu$ to the objective (for a given $\nu$). From the perspective of synthesis, these problems are most important when player 1 is restricted to finite-memory strategies (in Example 5 we show that infinite-memory strategies may yield a better value for player 1, hence the restriction to finite-memory strategies may affect the analysis of the synthesis problem).

For mean-payoff expressions, optimal finite-memory strategies may not always exist. Hence, the quantitative analysis problem is to compute the greatest lower bound on the minimal value that player 1 can assure. We note that since all model checking problems (i.e., the quantitative generalization of the emptiness, universality and language inclusion) are decidable for mean-payoff expression, then the computability of the quantitative analysis will give us an effective algorithm to synthesize $\epsilon$-optimal finite-memory strategies (as we can enumerate all finite-memory strategies), and if the boolean analysis problem were decidable, then we would have an algorithm that construct the corresponding player-1 strategy.

\textbf{Our contribution.} In this chapter, we consider for the first time the synthesis problem for a robust class of quantitative objectives, namely, for the class of mean-payoff expressions. We prove computability for the quantitative synthesis problem, and we show that the boolean analysis problem is inter-reducible with Hilbert’s tenth problem over rationals ($H10(\mathbb{Q})$), which is a fundamental long-standing open question in computer science and mathematics. We show that the problem is inter-reducible with $H10(\mathbb{Q})$ even when both players are restricted to finite-memory strategies, and we show that there is a fragment of mean-payoff expressions that is $H10(\mathbb{Q})$-hard when one or both players are restricted to finite-memory strategies, but decidable when both players may use infinite-memory strategies.

Our main technical contribution is the introduction of a general scheme that lifts a one-player game solution (equivalently, a model checking algorithm) to a solution for a
two-player game (when player 1 is restricted to finite-memory strategies). The scheme works for a large class of quantitative objectives that have certain properties (which we define in Subsection 5.1.2).

**Related work.** The class of mean-payoff expressions was introduced in [31], and the decidability of the model checking problems (which correspond to one-player games) was established. A simpler and more efficient algorithm for mean-payoff expression games was given in [96] (and in the previous chapter). Mean-payoff games on multidimensional graphs were first studied in [35]. In these games the objective of player 1 was to satisfy a conjunctive condition (in the terms of this chapter, the objective was a maximum of multiple one-dimensional objectives). In [99], decidability for an objective that is formed by the min and max operators was established. But the proof cannot be extended to include the numerical complement operator, and it does not scale for the case that player 1 is restricted to finite-memory strategies.

**Structure of the chapter.** In the next section we give the basic definitions for quantitative games and we define a class of quantitative objectives that have special properties. In Sections 5.2 and 5.3 we give a generic solution for the synthesis problem of quantitative objectives that satisfies the special properties (and an overview of the solution is given in Subsection 5.1.3). In Section 5.4 we show that mean-payoff expressions satisfy the special properties and the main results of the chapter follow.

**Bibliographic note.** The results of this chapter were first published in: Yaron Velner: Finite-memory strategy synthesis for robust multidimensional mean-payoff objectives. CSL-LICS 2014.

**5.1 Games with Quantitative Objectives**

In this section we give the formal definitions for quantitative objectives and games on graphs with quantitative objectives (Subsection 5.1.1). We define four special properties of quantitative objectives (Subsection 5.1.2), and we give an informal overview for the two-player game solution of games with quantitative objectives that satisfy the special properties (Subsection 5.1.3).

**5.1.1 Quantitative games on graphs**

**Quantitative objectives.** In this chapter we consider directed finite graphs with a k-dimensional weight function that assigns a vector of rationals to each edge. A *quantitative objective* is a function that assigns a value to every infinite sequence of weight vectors. Formally an objective is a function \( \text{obj} : (\mathbb{R}^k)^{\omega} \rightarrow \mathbb{R} \). A simple example for quantitative objective is to consider a one-dimensional weight function and an objective that assigns to each infinite path the maximal weight that occurs infinitely often in the path. An objective \( \text{obj} : (\mathbb{R}^k)^{\omega} \rightarrow \mathbb{R} \) is called *prefix-independent* if for every \( a_1 \in (\mathbb{R}^k)^* \) and \( a_2 \in (\mathbb{R}^k)^{\omega} \) it holds that \( \text{obj}(a_1 a_2) = \text{obj}(a_2) \).

**Strategies and finite-memory strategies.** A *strategy* is a recipe for determining the next move based on the *history* of the play. A *player-i strategy* is a func-
A strategy has finite memory if it can be implemented by a Moore machine \((M, m_0, \alpha_n, \alpha_u)\), where \(M\) is a finite set of memory states, \(m_0\) is the initial memory state, \(\alpha_u : M \times V \rightarrow M\) is the update function, and \(\alpha_n : M \times V_i \rightarrow V\) is the next vertex function. If a play prefix is in state \(v_i\) and memory state \(M\), then the strategy choice for the next vertex is \(v = \alpha_n(M, v_i)\) and the memory is updated to \(\alpha_u(M, v_i)\). A strategy is memoryless if it depends only in the current location of the pebble. Formally a player-\(i\) memoryless strategy is a function \(\sigma : V_i \rightarrow V\). (We note that a memoryless strategy is also a finite-memory strategy.)

We denote the set of all player-\(i\) strategies by \(\mathcal{S}_i\) and we denote the set of all player-\(i\) finite memory strategies by \(\mathcal{FM}_i\).

**Game graph according to a finite-memory strategy.** For a game graph \(G = (V = V_1 \cup V_2, E, w)\) and a player-1 finite-memory strategy \(\sigma = (M, m_0, \alpha_u, \alpha_n)\), we denote the game graph according to strategy \(\sigma\) by \(G^\sigma\), and we define it as follows:

- The vertices of \(G^\sigma\) are the Cartesian product \(V \times M\); player-\(i\) vertices are \(V_i \times M\); and the initial vertex of \(G^\sigma\) is \((v_0, m_0)\).

- For a player-1 vertex \((v, m)\), the only successor vertex is \((\alpha_n(v, m), \alpha_u(v, m))\).

- For a player-2 vertex \((v, m)\) the set of successor vertices is \(\{(u, n) \mid (v, u) \in E\ \text{ and } \alpha_u(v, m) = n\}\).

We note that the out-degree of all player-1 vertices is one, and thus \(G^\sigma\) is a one-player game graph. The main property of graphs according to a finite-memory strategy is that every infinite path in \(G^\sigma\) corresponds to a play that is consistent with \(\sigma\) in \(G\). A game graph according to a memoryless strategy is a special case of games according to finite-memory strategies. In this case, the game graph is obtained from \(G\) by removing all the player’s out-edges that are not chosen by the memoryless strategy.

**Example 2.** Consider the game graph in Figure 5.1 and consider a player-1 strategy \(\sigma\) that in vertex \(v_2\) moves the pebble to \(v_3\) if \(v_2\) was visited an odd number of times and otherwise it moves the pebble to \(v_1\). For example, in the first time that \(v_2\) is visited, player 1 moves the pebble to \(v_3\), in the second time he will move the pebble to \(v_1\), in the third time to \(v_3\) and so on. The strategy \(\sigma\) requires one bit of memory (i.e., \(M = \{0, 1\}\)), and \(G^\sigma\) is illustrated in Figure 5.2 (the labeling of the nodes represents the memory state). In \(G^\sigma\) all the choices are done by player 2.

**Values of strategies and games.** A tuple \((\sigma, \tau)\) of player-1 and player-2 strategies (respectively) uniquely defines a play \(\pi_{\sigma, \tau}\) in a given graph. For a game graph \(G\), a quantitative objective \(\text{obj}\) and a tuple of strategies \((\sigma, \tau)\) we denote \(\text{Val}_{\sigma, \tau} = \text{obj}(\pi_{\sigma, \tau})\). In this chapter, we assume that player 1 wishes to minimize\(^1\) the value of the quantitative objective, and we define the value of a player-1 strategy \(\sigma\) to be \(\text{Val}_\sigma = \sup_{\tau \in \mathcal{S}_2} \text{Val}_{\sigma, \tau}\).

\(^1\)Since we consider robust objectives, then the same results hold when player-1 goal is to maximize the value of the objective.
CHAPTER 5. FINITE-MEMORY STRATEGY SYNTHESIS FOR
MULTIDIMENSIONAL MEAN-PAYOFF OBJECTIVES

Figure 5.1: Game graph $G$. Player 1 owns the round vertices.

Figure 5.2: Game graph $G$ according to strategy $\sigma$.

(Intuitively, this is the maximal value that player 2 can achieve against strategy $\sigma$.) The minimal value of a game is defined as $\inf_{\sigma \in \mathcal{FM}_1} \text{Val}_\sigma$. Intuitively, the minimal value of a game is the minimal value that player 1 can ensure by a finite-memory strategy.

Quantitative and boolean analysis For a given game graph, objective, and a rational threshold $r \in \mathbb{Q}$: The quantitative analysis task is to compute the minimal value of the game that can be enforced by a finite-memory strategy. The boolean analysis task is to decide whether there is a player-1 finite-memory strategy $\sigma$ for which $\text{Val}_\sigma \leq r$. That is, whether player 1 can assure a value of at most $r$ for the objective.

Boolean games and winning strategies. A boolean game is a game on graph equipped with a winning condition $W \subseteq E^\omega$ (that is, a winning condition is a set of infinite paths). A play $\pi$ is winning for player 1 if $\pi \in W$, and a strategy $\sigma$ is a player-1 winning strategy if for every player-2 strategy $\tau$ we have $\pi_{\sigma,\tau} \in W$. For a quantitative objective $\text{obj}$ and a threshold $\nu \in \mathbb{R}$ we denote by $(\text{obj}, \nu)$ the boolean winning condition $\{\pi \in E^\omega \mid \text{obj}(\pi) \leq \nu\}$.

5.1.2 One-player game solution

In this chapter, we consider objectives that have special properties for their one-player game solution and we present a general scheme that lifts a one-player game solution into a two-player game solution. To formally define the special properties of the solutions, we give the next definitions.

Definitions and notions for weighted graphs. Let $G = (V, E, w : E \to \mathbb{Q}^k)$ be a $k$-dimensional weighted graph. The weight vector of a finite path $\pi = e_1 \ldots e_n$ is $w(\pi) = \sum_{i=1}^n w(e_i)$ and the average weight of a path is $\text{Avg}(\pi) = \frac{w(\pi)}{|\pi|}$. For a set of
finite paths $\Pi = \{\pi_1, \ldots, \pi_n\}$ we denote $\text{Avg}(\Pi) = \{\text{Avg}(\pi_1), \ldots, \text{Avg}(\pi_n)\}$. We denote the set of simple cycles in $G$ by $C(G)$, and we abbreviate $\text{Avg}(G) = \text{Avg}(C(G))$. For a finite set of vectors $V = \{v_1, v_2, \ldots, v_n\} \in \mathbb{R}^k$, we denote $\text{CONVEX}(V) = \{\sum_{i=1}^n \alpha_i v_i \mid \sum_{i=1}^n \alpha_i = 1 \text{ and } \alpha_1, \ldots, \alpha_n \geq 0\}$ (see Figure 5.3). We abbreviate $\text{CONVEX}(G) = \text{CONVEX}(\text{Avg}(G))$. An $m$-dimensional simplex is the set $S(m) = \{(x_1, \ldots, x_m) \in \mathbb{R}^m \mid x_i \geq 0 \land \sum_{i=1}^m x_i = 1\}$. The simplex interior is $SI(m) = \{(x_1, \ldots, x_m) \in \mathbb{R}^m \mid x_i > 0 \land \sum_{i=1}^m x_i = 1\}$, and the rational interior of a simplex is $QSI(m) = SI(m) \cap \mathbb{Q}^m$. When $m$ is clear from the context we abbreviate $S(m), SI(m)$ and $QSI(m)$ with $S, SI$ and $QSI$ (respectively).

Solution for one-player game with special properties. A solution for a one-player quantitative game is a function $f$ that assigns to every one-player game graph $G$ the maximal value that the player can achieve in graph $G$. We note that for prefix-independent objectives, a function $f'$ that assigns to each strongly connected graph its maximal value uniquely defines the solution function $f$ (since the value of $f$ is the maximal value of $f'$ over all the strongly connected component of the graph). In this chapter, we will consider only prefix-independent objective, hence, we define the special properties of a solution for strongly connected graphs. The special properties that we consider are:

1. First-order definable. For every $n \in \mathbb{N}$ there is a first-order formula $\zeta_n(x_1, \ldots, x_n, y)$ over $\langle \mathbb{R}, =, <, +, \times \rangle$ such that for every graph $G$ with $\text{Avg}(G) = \{x_1, \ldots, x_n\}$ we have $f(G) = y$ if and only if $\zeta_n(x_1, \ldots, x_n, y)$ holds. In addition we require $\zeta_n$ to be computable from $n$. In the sequel, we write $y = \zeta_n(x_1, \ldots, x_n)$ instead of $\zeta_n(x_1, \ldots, x_n, y)$.

2. Monotone in $\text{CONVEX}(G)$. If for two (strongly connected) graphs $H$ and $G$ we have $\text{CONVEX}(G) \subseteq \text{CONVEX}(H)$, then $f(G) \leq f(H)$. As a consequence, we get that for a $k$-dimensional objective, $f$ is a function from $\mathbb{R}^k$ to $\mathbb{R}$, and that $f(G) \equiv g(\text{CONVEX}(G))$ for some function $g : (\mathbb{R}^k)^* \rightarrow \mathbb{R}$. Hence, by abusing the notation, we sometime write $f(\text{CONVEX}(G))$ instead of $f(G)$.

3. Continuous function. $f$ is a continuous function. Formally, if $f$ is the solution for a $k$-dimensional objective, then for every $n \in \mathbb{N}$ the function $\zeta_n : (\mathbb{R}^k)^n \rightarrow \mathbb{R}$ is a continuous function, i.e., for every $\epsilon > 0$ there exists $\delta > 0$ such that for every two vectors $A, B \in (\mathbb{R}^k)^n$ with $|A - B| < \delta$ it holds that $|\zeta_n(A) - \zeta_n(B)| < \epsilon$.  

Figure 5.3: $\text{CONVEX}(A, B, C, D, E, F, G)$ is the polygon $ABDEFG$.  

![Diagram of CONVEX polygon ABDEFG]
We will show computability for the quantitative analysis problem for objectives that have a solution that satisfies the above three properties. We also consider a fourth special property, and we will show decidability for the boolean analysis problem for objectives that have a solution that satisfies all four properties.

4. **Fourth property.** A solution \( f = \{ \zeta_1, \ldots, \zeta_n, \zeta_{n+1}, \ldots \} \) satisfies the fourth property if the next problem is decidable (for the set \( \{ \zeta_1, \ldots, \zeta_n, \zeta_{n+1}, \ldots \} \)):

- **Input:** a threshold \( \nu \in \mathbb{Q} \) and a set of \( n \) matrices \( A_1, \ldots, A_n \), where \( A_i \) is a \( k \times m_i \) matrix for some \( m_i \in \mathbb{N} \).
- **Task:** determine if the inequality \( \zeta_n(A_1 \cdot x_1, \ldots, A_n \cdot x_m) \leq \nu \) subject to \( x_i \in \mathcal{QSI}(m_i) \) is feasible (note that the result of the multiplication \( A_i \cdot x_i \) is a vector of size \( k \)).

In the next example we demonstrate the above properties.

**Example 3.** Consider the two-dimensional one-player solution function \( f(G) = \max_{(x,y)\in \text{CONVEX}(G)}[\max(x+y+10, -x+y+10, \min(-x+y-10, x+y-10))]. \) We demonstrate that \( f \) is first-order definable by giving the explicit formula for \( \zeta_2 \), that is, the formula for a (strongly connected) graph with only two simple cycles with average weights \((x_1, y_1)\) and \((x_2, y_2)\).

\[
\begin{align*}
\zeta_2(x_1, y_1, x_2, y_2, r) &\equiv \\
& \forall \alpha_1, \alpha_2, x, y (\alpha_1 \geq 0 \land \alpha_2 \geq 0 \land (\alpha_1 + \alpha_2 = 1) \land (x = \alpha_1 x_1 + \alpha_2 x_2) \land (y = \alpha_1 y_1 + \alpha_2 y_2)) \rightarrow \\
& r \geq \max(x+y+10, -x+y+10, \min(-x+y-10, x+y-10)) \land \\
& \exists \alpha_1, \alpha_2, x, y (\alpha_1 \geq 0 \land \alpha_2 \geq 0 \land (\alpha_1 + \alpha_2 = 1) \land (x = \alpha_1 x_1 + \alpha_2 x_2) \land (y = \alpha_1 y_1 + \alpha_2 y_2)) \land \\
& r = \max(x+y+10, -x+y+10, \min(-x+y-10, x+y-10))
\end{align*}
\]

(Technically max and min are not in \( \langle \mathbb{R}, <, +, \times \rangle \), but they are trivially definable in this vocabulary.) Clearly if for two graphs we have \( \text{CONVEX}(G_1) \subseteq \text{CONVEX}(G_2) \), then \( f(G_1) \leq f(G_2) \) (hence, \( f \) is monotone), and \( \zeta_2 \) is obviously a continuous function (and in general \( \zeta_n \) is also continuous). Hence, \( f \) satisfies Properties 1-3. In Figure 5.4 we illustrate the geometrical interpretation of Property 2, namely, the fact that the value of \( f \) depends only in \( \text{CONVEX}(G) \). The equality \( \max(x+y+10, -x+y+10, \min(-x+y-10, x+y-10)) = 0 \) is represented by the thick line. The points that are connected by the dotted line represent the weights of the simple cycles of a strongly connected graph \( G_1 \) and the points that are connected by the dashed line represent the weights of the simple cycles of a strongly connected graph \( G_2 \). The reader can see that \( \text{CONVEX}(G_1) \) is below the thick line and \( \text{CONVEX}(G_2) \) intersects with it. Hence, \( f(G_1) < 0 \) and \( f(G_2) > 0 \).
5.1.3 Informal overview of the solution for two-player games

The key notion for our solution is games according to strategies. When a one-player solution $f$ is given, the boolean analysis problem amounts to determining whether there is a finite-memory strategy $\sigma$ such that for every strongly-connected component (SCC) $S \in G^\sigma$ it holds that $f(\text{CONVEX}(S)) \leq \nu$. In Lemma 22 we show that w.l.o.g we may assume that for any $\sigma$ the graph $G^\sigma$ is strongly connected. Hence, in Section 5.2 we investigate the set $\{\text{CONVEX}(G^\sigma) \mid \sigma \in \mathcal{FM}_1\}$ and obtain a computable characterization for it. In Section 5.3 we exploit the properties of the one-player solution and the results of Section 5.2 and we obtain a first-order formula over rationals that computes the values that player 1 can enforce. We use the fact that $f$ is continuous to show that the formula has the same infimum over rationals and reals, and hence, due to Tarski’s Theorem the infimum value is computable. We also show that if Property 4 holds, then one can effectively determine whether the formula has an assignment that gives a value of at most $\nu$. In Section 5.4 we apply these results for mean-payoff expressions. We show that their one-player solution satisfies Properties 1-3, and that it satisfies property 4 if and only if $H10(\mathbb{Q})$ is decidable.

5.2 CONVEX Cycles Problem

In this section, we consider the next problem:

**Problem 1** (CONVEX cycles problem).

- Input: a $k$-dimensional game graph $G$ and a set of $k$-dimensional vectors $V$.

- Task: determine whether there is a player-1 finite-memory strategy $\sigma$ such that $\text{CONVEX}(G^\sigma) \subseteq \text{CONVEX}(V)$. (We call such strategy a realizing strategy.)

We first present the solution (namely, an algorithm) for the above problem, and then we show how to find all the sets of vectors for which there is a realizing player-1 finite-memory strategy.

The solution for Problem 1 relies on the next lemma.

**Lemma 19.** For a game graph $G$ and a set of vectors $V$, there exists a player-1 finite-memory strategy $\sigma$ for which $\text{CONVEX}(G^\sigma) \subseteq \text{CONVEX}(V)$ iff for every player-
Proof. The proof for the direction from left to right is trivial (since any cycle in \((G^\lambda)^\sigma\) is also a cycle in \(G^\sigma\)).

Our proof for the converse direction is inspired by [63], and the key intuition of the proof is the following. Let \(v\) be a player-2 vertex, with two out-edges \(e_1\) and \(e_2\), and let \(G_1 = G - \{e_1\}\) and \(G_2 = G - \{e_2\}\). Suppose that player 1 has two finite-memory strategies \(\sigma_1\) and \(\sigma_2\) such that \(\text{CONVEX}(G_1^{\sigma_i}) \subseteq \text{CONVEX}(V)\) (for \(i = 1, 2\)). Then player 1 can combine the two strategies over \(G - \{e_1\}\) and \(G - \{e_2\}\), and he can obtain a finite-memory strategy \(\sigma\), such that each simple cycles in \(G^\sigma\) is a convex combination of cycles from \(G_1^{\sigma_1}\) and \(G_2^{\sigma_2}\), and hence, \(\text{CONVEX}(G^\sigma) \subseteq \text{CONVEX}(V)\). Hence, either player 1 has a realizable strategy for \(G\) or he does not have realizable strategy for \(G_1 = G - \{e_1\}\) or for \(G_2 = G - \{e_2\}\). Since this holds for every player-2 state, the proof follows.

In order to formally prove the key intuition we claim that if player 1 has a realizable strategy \(\sigma_i\) against any player-2 memoryless strategy, then he has a realizable strategy \(\sigma\) that satisfies \(\text{CONVEX}(G^\sigma) \subseteq \text{CONVEX}(V)\), and we prove the claim by induction on the number of player-2 vertices with out-degree greater then one. The base case, where all of player-2 vertices have out-degree one, is trivial. For the inductive step, let us assume that there is a player-2 vertex \(v\) with out-edges \(e_1\) and \(e_2\) (if there is no such vertex, then we are in the base case). For \(i = 1, 2\), let \(G_i = G - \{e_i\}\). If player-2 has a violating memoryless strategy in either \(G_1\) or \(G_2\), then surely this is also a violating memoryless strategy for \(G\), and the claim follows. Otherwise, we construct a realizable player-1 strategy in \(G\) in the following way. For \(i = 1, 2\), let \(\sigma_i\) be a finite-memory player-1 realizable strategy in \(G_i\). If in \(\sigma_1\) (resp., \(\sigma_2\)), the vertex \(v\) is unreachable then it is surely a winning strategy also for \(G\). Otherwise, there exists a memory state \(m\) such that \((m, v)\) is a vertex in \(G_1^{\sigma_1}\), and we denote by \(\sigma_i'\) the strategy that is obtained by changing \(\sigma_1\) initial memory state to \(m\). We construct \(\sigma\) in the following way. The memory structure of \(\sigma\) is a tuple \((M_1, M_2, \{1, 2\})\) where \(M_1\) is the memory structure of \(\sigma_1'\), \(M_2\) is the memory structure of \(\sigma_2\), and the third value in the tuple indicates if we are playing according to \(\sigma_1'\) or \(\sigma_2\). At the beginning of a play, \(\sigma\) decides according to \(\sigma_2\) (and updates \(M_2\) accordingly). If \(\sigma\) decides according to \(\sigma_2\) and edge \(e_1\) is visited, then \(\sigma\) decides according to \(\sigma_1'\) (and updates \(M_1\) accordingly), until edge \(e_2\) is visited, and then \(\sigma\) again decides according to \(\sigma_2\), and so on. We note that \(\sigma\) is a finite-memory strategy, and that any simple cycle in \(G^\sigma\) is a composition of simple cycles from \(G_1^{\sigma_1}\) and \(G_2^{\sigma_2}\). Hence, the average weight of any simple cycle in \(G^\sigma\) is \(\lambda x_1 + (1 - \lambda)x_2\) for some \(\lambda \in [0, 1]\) and \(x_i \in \text{Avg}(G_i) \subseteq \text{CONVEX}(V)\). And thus, a convex combination of \(x_1\) and \(x_2\) is also in \(\text{CONVEX}(V)\), and we get that \(\text{CONVEX}(G^\sigma) \subseteq \text{CONVEX}(V)\). Therefore, \(\sigma\) is a realizing strategy and the proof is completed. 

\(\square\)
We now wish to characterize all the sets of vectors that have a realizing strategy. For this purpose we give the next definition. For a player-2 memoryless strategy \( \tau \), let \( \Pi_\tau^e \) be the (finite) set of Eulerian cyclic paths in \( G^\tau \), that is \( \Pi_\tau^e \) contains only cyclic paths that visit every edge at most once. For every path \( \pi \in \Pi_\tau^e \), let \( c_1, \ldots, c_\ell \) be the simple cycles that occur in \( \pi \) and we associate a \( t \times k \) matrix \( A_\pi \) to every path \( \pi \) such that the \( i \)-th column of the matrix is \( \text{Avg}(c_i) \). We observe that

\[
\{ \text{Avg}(\pi) \mid \pi \text{ is a cyclic path in } G^\tau \} = \bigcup_{\pi \in \Pi_\tau^e} \{ A_\pi \cdot x \mid x \in \mathbb{QSI} \}
\]

The next lemma shows how to compute the realizable sets of vectors.

Lemma 20. Let \( G \) be a \( k \)-dimensional graph, and let \( \tau_1, \ldots, \tau_\ell \) be the (finitely many) player-2 memoryless strategies in \( G \). A set of vectors \( V \subseteq \mathbb{R}^k \) is realizable if and only if there exist \( x_1, \ldots, x_\ell \in \mathbb{R}^k \) such that \( x_i \in \bigcup_{\pi \in \Pi_\tau^e} \{ A_\pi \cdot x \mid x \in \mathbb{QSI} \} \) (for every \( i \in \{1, \ldots, \ell\} \)) and \( \text{CONVEX}(x_1, \ldots, x_\ell) \subseteq \text{CONVEX}(V) \).

Proof. First we characterize the realizable vector sets when a player-2 memoryless strategy \( \tau \) is given, that is, we characterize the realizable vectors in a one-player game. A finite-memory strategy \( \sigma \) in a one-player graph \( G^\tau \) is an ultimately periodic infinite path, and \( (G^\tau)^\sigma \) is a lasso shaped graph with exactly one cycle. The cycle of \( (G^\tau)^\sigma \) is obviously a cyclic path in \( G^\tau \), and thus \( V \) is realizable in \( G^\tau \) iff there is a cyclic path \( \pi \) in \( G^\tau \) with \( \text{Avg}(\pi) \in \text{CONVEX}(V) \).

Hence, by Lemma 19, we get that \( V \) is realizable iff for every player-2 memoryless strategy \( \tau_i \) there is a cyclic path \( \pi_i \) in \( G^\tau \) with \( \text{Avg}(\pi_i) \in \text{CONVEX}(V) \). Since such witness \( \pi_i \) exists iff there exists \( x_i \in \bigcup_{\pi \in \Pi_\tau^e} \{ A_\pi \cdot x \mid x \in \mathbb{QSI} \} \) with \( \text{Avg}(\pi_i) = x_i \), then the proof is completed.

In the next example we illustrate the geometrical interpretation of Lemma 20.

Example 4. Consider the game graph \( G \) in Figure 5.5, where the box vertices are controlled by player 2. Player 2 has two possible memoryless strategies, namely, \( \tau_1 \) that follows the edge \( v_0 \to v_1 \) and \( \tau_2 \) that follows \( v_0 \to v_4 \). In \( G^{\tau_1} \) the set of Eulerian cyclic paths \( \Pi_\tau^{e_1} \) contains all cyclic sub-paths of the Eulerian cyclic path \( v_1 \to v_2 \to v_2 \to v_3 \to v_3 \to v_1 \). Hence, the average weight of any infinite lasso path in \( G^{\tau_1} \) is a convex combination of \( \text{Avg}(v_1 \to v_2 \to v_3 \to v_1) \), \( \text{Avg}(v_2 \to v_2) \) and \( \text{Avg}(v_3 \to v_3) \) (points \( D, F \) and \( E \) in Figure 5.6). In \( G^{\tau_2} \), an Eulerian cyclic path is either a sub-path of \( v_4 \to v_5 \to v_5 \to v_4 \) or the path \( v_6 \to v_6 \). Hence, the average weight of any infinite lasso path is either a convex combination of \( \text{Avg}(v_4 \to v_5 \to v_4) \) and \( \text{Avg}(v_5 \to v_5) \) (points \( A \) and \( B \) in Figure 5.6), or it is \( \text{Avg}(v_6 \to v_6) \) (point \( C \) in Figure 5.6). By Lemma 20, we get that a set of vectors \( V \) is realizable if and only if \( \text{CONVEX}(V) \) intersects with the polygon \( DEF \) and with either the line \( AB \) or with the point \( C \) (or with both).
5.3 Generic Solution for Games with Quantitative Objectives

In this section we solve the quantitative analysis problem for games with quantitative objectives that satisfy Properties 1-3 and we solve the boolean analysis problem for objectives that satisfy Properties 1-4. We first give a conceptual (i.e., not always computable) algorithm (or a scheme) for the boolean analysis problem, and then extend the algorithm for the quantitative analysis problem.

An equivalent formulation for the boolean analysis problem is to ask whether for a game graph $G$ and a threshold $\nu$ there is a player-1 (finite-memory) strategy $\sigma$ such that the one-player solution over $G^\sigma$ is at most $\nu$. By the third property (convex monotonicity), it is enough to determine whether there is $\sigma$ such that for every SCC $S$ of $G^\sigma$ it holds that $f(CONVEX(S)) \leq \nu$ (where $f$ is the solution for the one-player game). However, we first show how to determine whether there is $\sigma$ such that $f(CONVEX(G^\sigma)) \leq \nu$ and only then solve the original problem.

**Lemma 21.** Let $f$ be a one-player solution that satisfies Properties 1-3. Then $\inf_{\sigma \in FM_1} f(CONVEX(G^\sigma))$ is computable (when the input is a game graph $G$). If $f$ also satisfies Property 4, then the problem of determining whether there is a player-1 finite-memory strategy $\sigma$ such that $f(CONVEX(G^\sigma)) \leq \nu$ is decidable (when the input is $G$ and $\nu$).

**Proof.** Let $\tau_1, \ldots, \tau_m$ be all player-2 memoryless strategies in $G$ (note that $m$ is at most exponential in $|G|$). By Lemma 20, and by the monotonicity of $f$, there is a player-1 strategy $\sigma$ that satisfies $f(CONVEX(G^\sigma)) \leq \nu$ if and only if there are matrices $A_{\pi_1}, \ldots, A_{\pi_m}$ and vectors $x_1, \ldots, x_m$ such that $\pi_i \in \Pi_{FM}^0$ and $x_i \in QST$ and
computes player-1 winning region, and we prove its correctness in
Lemma 4. In $\Pi^e$, every player-1 vertex with out-degree 0 belongs to the attractor. Moreover, every player-1 vertex
is in the player-1 attractor. Hence, for every SCC $S$ in $H$ such that $f(\text{CONVEX}(S)) \leq \nu$. Since
CONVEX($S$) $\subseteq$ CONVEX($S'$) and by the monotonicity of $f$ we get that $f(\text{CONVEX}(S)) \leq \nu$ and therefore $\sigma$ is a winning strategy in $(H, u)$.

Before presenting the algorithm for the boolean analysis problem we recall the
(standard) definitions of winning regions and attractors. Let $G$ be a game graph with
an initial vertex $v_0$, and let $v$ be an arbitrary vertex in $G$. We denote by $(G, v)$ the
game graph that is formed from $G$ by changing the initial vertex to $v$. We say that
a vertex $v$ is in player-1 winning region (denoted by $\text{Win}_1$) if player 1 wins in $(G, v)$
(that is, player 1 has a finite-memory strategy that assures a value at most $\nu$ to the
objective). The player-1 attractor set of a vertex $v$ (denoted by $\text{Attr}_1(v)$) contains all
the vertices from which player 1 can force reachability to $v$ (after finite number
of rounds). It is well known that the attractor set of a vertex is computable (even
in linear time) and that player 1 can force reachability by a finite-memory strategy
(in fact, even by a memoryless strategy). The next remark shows another important
property of attractors and winning regions.

Remark 4. Let $G$ be a game graph over a boolean objective that is formed by a quanti-
titative objective with a solution function $f$ and a threshold $\nu$, and let $v$ be a vertex in
$G$. Then for every vertex $u \notin \text{Attr}_1(v)$, if $\sigma$ is a finite-memory player-1 strategy for
$(G, u)$, then $\sigma$ is a winning strategy in $(G - \text{Attr}_1(v), u)$.

Proof. We denote $H = G - \text{Attr}_1(v)$ and we observe that $(H, u)^\sigma$ is a subgraph of
$(G, u)^\sigma$ and that the out-degree if every vertex in $H$ is at least one (by definition a
vertex with out-degree 0 belongs to the attractor). Moreover, every player-1 vertex
in $H$ does not have a transition to $\text{Attr}_1(v)$ in $G$ (otherwise the vertex belongs to the
attractor). Hence, for every SCC $S$ in $H$ there is a corresponding SCC $S'$ in $G$ such that
$f(\text{CONVEX}(S')) \leq \nu$. Since $\text{CONVEX}(S) \subseteq \text{CONVEX}(S')$ and by the monotonicity
of $f$ we get that $f(\text{CONVEX}(S)) \leq \nu$ and therefore $\sigma$ is a winning strategy in $(H, u)$.

Algorithm 1 computes player-1 winning region, and we prove its correctness in
Lemma 22.
\textbf{Algorithm 1} Player-1 winning region computation for quantitative objectives
\begin{verbatim}
WINNINGREGION(G, f, ν)
    for v ∈ G do
        if ∃σ s.t. f(CONVEX((G, v)σ)) ≤ ν then
            W ← Attr₁(v)
            W ← W ∪ WINNINGREGION(G − Attr₁(v), f, ν)
        return W
    end for
    return ∅
\end{verbatim}

\textbf{Lemma 22.} \textit{Algorithm 1 computes player-1 winning region.}

\textit{Proof.} We first prove that in every step of the algorithm, if a vertex \( u \in W \), then \( u \in \text{Win}_1 \). We prove the assertion by considering the next three cases: (i) There is a strategy \( σ \) for which \( f(\text{CONVEX}((G, u)^σ)) \leq ν \). In this case, for every SCC \( S \in (G, u)^σ \) we have that \( \text{CONVEX}(S) \subseteq \text{CONVEX}((G, u)^σ) \) and by the monotonicity of \( f \) we get that \( f(\text{CONVEX}(S)) \leq ν \). Hence, \( u \in \text{Win}_1 \). (ii) There is a vertex \( v \) and a strategy \( σ \) s.t. \( f(\text{CONVEX}((G, v)^σ)) \leq ν \) and \( u \in \text{Attr}_1(v) \). In this case, \( v \) is in player-1 winning region and therefore the attractor of \( v \) is also in \( \text{Win}_1 \). (iii) For some vertex \( v \) we have \( u \in \text{WINNINGREGION}(G − \text{Attr}_1(v), f, ν) \) and \( f(\text{CONVEX}((G, v)^σ)) \leq ν \) for some strategy \( σ \). By a simple induction on the size of the graph we get that \( u \) is in player-1 winning region for the game graph \( G − \text{Attr}_1(v) \). The following strategy is a winning strategy for \( (G, u) \): (a) play according to the winning strategy over \( G − \text{Attr}_1(v) \); (b) if the pebble is in vertex \( v \), then play according to \( σ \). Hence, if \( u \in W \), then \( u \in \text{Win}_1 \) and we get that \( \text{Win}_1 \supseteq W \).

In order prove the converse direction, we first prove that if \( \text{Win}_1 \neq ∅ \), then \( W \neq ∅ \). Indeed, if \( v \in \text{Win}_1 \), then for some strategy \( σ \) we have that for every SCC \( S \in (G, v)^σ \) it holds that \( f(\text{CONVEX}(S)) \leq ν \). Let \( S' \) be a terminal SCC in \( (G, v)^σ \) and let \((u, m)\) be a vertex in \( S' \) (where \( u \) is a vertex in \( G \) and \( m \) is a memory state of \( σ \)). Let \( σ' \) be the strategy that is formed by changing \( σ \) initial memory state to \( m \). Then \( (G, u)^{σ'} = S' \), and therefore \( f(\text{CONVEX}((G, u)^{σ'})) \leq ν \). Hence, the \textbf{if} condition in the \textbf{for} loop is satisfied at least once, and \( W \neq ∅ \). We are now ready to prove that \( \text{Win}_1 \subseteq W \). Towards a contradiction we assume the existence of \( u \in (\text{Win}_1 − W) \). By the definition of Algorithm 1 it follows that there is a subgraph \( H \subseteq G \) such that \( u \in H \) and the algorithm returns \( ∅ \) when it runs over \( H \). Hence, player-1 winning region in \( H \) is empty (namely, \( u \notin \text{Win}_1 \) over game graph \( H \)) and by Remark 4 we get that \( u \notin \text{Win}_1 \) in game graph \( G \) and the contradiction follows. Thus \( \text{Win}_1 \subseteq W \).

We present a similar algorithm for the computation of quantitative analysis of quantitative objectives. For this purpose we extend the notion of winning regions to quantitative objectives by defining \textit{value regions}. For a threshold \( ν \) we say that
a vertex $v$ is in $\nu$ value region (denoted by $VR(\nu)$) if $\inf_{\sigma \in \mathcal{F}M_1} \sup_{\tau \in S_2} Val_{\sigma,\tau} = \nu$ (when the initial vertex of the game is $v$). Algorithm 2 computes value regions by a call to $VALUE_REGION(G, f, -\infty)$, and its correctness follows by the same arguments as in the proof of Lemma 22. We note that if $f$ satisfies Properties 1-3, then by

**Algorithm 2** Value region computation for quantitative objectives. The algorithm invokes $VALUE_REGION(G, f, -\infty)$.

```
VALUE_REGION(G, f, ValLowerBound)
if G ≠ ∅ then
for v ∈ G do
   I[v] ← max(\(\inf_{\sigma \in \mathcal{F}M_1} f(CONVEX((G,v)^\sigma))\), ValLowerBound)
end for
u ← argmin\(v \in G\) I[v] \{Choose u s.t I[u] = \(\min_{v \in G} I[v]\)\}
VR(I[u]) ← VR(I[u]) ∪ Attr1(u) \{Add Attr1(u) to the value region of I[u]\}
return VALUE_REGION(G − Attr1(u), f, I[u]) \{Continue the computation recursively. The new lower bound is I[u].\}
end if
```

Lemma 21, there is an effective procedure to compute $\inf_{\sigma \in \mathcal{F}M_1} f(CONVEX((G,v)^\sigma))$ (hence, Algorithm 2 can be effectively executed) and if $f$ satisfies Properties 1-4, then by the same lemma we get that there is a procedure to determine whether $f(CONVEX((G,v)^\sigma)) \leq \nu$ (hence, Algorithm 1 can be effectively executed). Hence, we get the main result of this section.

**Theorem 5.** Let $f$ be the one-player solution of a quantitative objective.

- If $f$ satisfies Properties 1-3, then the corresponding quantitative analysis problem is computable.
- If $f$ satisfies Properties 1-4, then the corresponding boolean analysis problem is decidable.

We note that Theorem 5 provides a recipe for the construction of $\epsilon$-optimal strategies. If the infimum value that player 1 can achieve is $\nu$, then the process that enumerates all $\sigma \in \mathcal{F}M_1$ and halts if the one-player solution of $G^\sigma$ is at most $\nu + \epsilon$ will always terminate. Similarly, if the boolean analysis problem is decidable, then it is possible to effectively construct a finite-memory strategy that assures the corresponding threshold (we first check if such a strategy exists, and if it does exist, then we enumerate all finite-memory strategies until we find a strategy $\sigma$ such that the solution for $G^\sigma$ is at most $\nu$).

### 5.4 Games with Mean-Payoff Expression Objectives

In this section, we use the results of Section 5.3 to analyze games with mean-payoff expressions. In Subsection 5.4.1 we introduce a normal form for mean-payoff expressions and show that optimal strategies may require infinite memory. In Subsection 5.4.2 we analyze mean-payoff expression games.
CHAPTER 5. FINITE-MEMORY STRATEGY SYNTHESIS FOR
MULTIDIMENSIONAL MEAN-PAYOFF OBJECTIVES

5.4.1 Mean-payoff expression objectives

We recall that the class of mean-payoff expressions is the closure of single dimension mean-payoff objectives under the algebraic operations of min, max, sum and numerical complement. For example, a possible expression is \( E = \text{min}(\text{LimInfAvg}_1, \text{LimSupAvg}_1 + \text{LimInfAvg}_2) + \text{max}(\text{LimInfAvg}_1, \text{LimSupAvg}_2) \), and the value of \( E \) for the sequence \((-1, 1)^w\) is \( \text{min}(-1, -1 + 1) + \text{max}(-1, 1) = 0 \).

We say that an expression \( E \) is of normal form if (i) the numerical complement does not occur in \( E \); and (ii) for every dimension \( i \), there is at most one occurrence of an atomic expression \( A_i \in \{ \text{LimInfAvg}_i, \text{LimSupAvg}_i \} \); and (iii) \( E = \text{max}(E_1, \ldots, E_\ell) \), where \( E_i \) is a max-free expression (that is, the max operator does not occur in \( E_i \)).

The next simple lemma shows that w.l.o.g we may consider only games over normal form expressions.

**Lemma 23.** For every \( k \)-dimensional weighted graph \( G \) with a weight function \( w \) and an expression \( E \), we can effectively construct an \( m \)-dimensional weight function \( w' \) and a normal form expression \( F \) such that every infinite path in \( G \) gets the same value according to \((E, w)\) and according to \((F, w')\).

**Proof.** We can easily overcome the restriction on the number of atomic expressions per dimension by creating several copies of the same dimension (that is, additional dimensions with weights that are identical to the original dimension). We can create an equivalent numerical complement free expression by the following recursive process. If \( E = -\text{LimInfAvg}_i \) (respectively, \( E = -\text{LimSupAvg}_i \)), then we multiply all the weights in dimension \( i \) by \(-1\) and define \( F = \text{LimSupAvg}_i \) (resp. \( F = \text{LimInfAvg}_i \)). \( F \) is equivalent to \( E \) since \( \text{LimInfAvg}(a_1, a_2, \ldots) = -\text{LimSupAvg}(-a_1, -a_2, \ldots) \). If \( E = -\text{op}(E_1, E_2) \), then we recursively change the weights and construct normal form expressions \( F_1 \) and \( F_2 \) that are equivalent to \(-E_1\) and \(-E_2\), and return the normal form expression \( F = \text{op}(F_1, F_2) \). And we similarly handle the expression \( E = \text{op}(E_1, E_2) \). Finally, if we have a numerical complement free expression \( E \), then we construct an equivalent expression \( F = \text{max}(F_1, \ldots, F_\ell) \), where \( F_i \) is a max free expression, by the following recursive procedure: If \( E \) is an atomic expression, then we return \( F = \text{max}(E, E) \). If \( E = \text{op}(E_1, E_2) \), then we recursively construct two expressions \( F_1 \) and \( F_2 \), such that \( F_1 \) is equivalent to \( E_i \) and \( F_1 = \text{max}(G_1, \ldots, G_r) \), \( F_2 = \text{max}(H_1, \ldots, H_q) \) (where \( H_i \) and \( G_i \) are max-free expressions), and we return \( F = \text{max}_{i \in \{1, \ldots, r\}, j \in \{1, \ldots, q\}} \{ \text{op}(G_i, H_j) \} \).

Hence, in the rest of the chapter we will assume w.l.o.g that all the expressions are in normal form. The next example shows that optimal strategies for mean-payoff expressions may require infinite memory.

**Example 5.** Consider the game graph in Figure 5.7 and the expression \( E = \text{max}(\text{LimInfAvg}_1, \text{LimInfAvg}_2) \). In this game graph there is only one vertex that is controlled by player 1 and two self-loop edges, namely \( e_1 \) with \( w(e_1) = (9, 1) \) and \( e_2 \) with \( w(e_2) = (1, 9) \). We first observe that any finite-memory strategy gives a value
of at least 5 to E. Indeed, a finite-memory strategy induces an ultimately periodic path \( \pi \) with \( \operatorname{LimInfAvg}(\pi) = \alpha w(e_1) + (1 - \alpha)w(e_2) \) for some \( \alpha \in [0, 1] \cap \mathbb{Q} \). Hence, \( E(\pi) = \max(9\alpha + 1 - \alpha, \alpha + 9 - 9\alpha) = \max(8\alpha + 1, 9 - 8\alpha), \) and the minimum value for \( E \) is obtained when \( \alpha = \frac{1}{2} \) and we get that the minimal value for \( E \) is 5. We now describe a player-1 infinite-memory strategy that gives a value of at most 2 to \( E \). The strategy is simple. It follows \( e_2 \) as long as the average weight in the first dimension is more than 2, then it follows \( e_1 \) as long as the average weight in the second dimension is more than 2, and this process is repeated forever (i.e., \( e_2 \) is followed for a while, then \( e_1 \) and so on). Clearly, in the formed path \( \pi \) the average weight of the first dimension is at most 2 for infinitely many prefixes of \( \pi \). Hence \( \operatorname{LimInfAvg}_1(\pi) \leq 2 \), and by the same arguments \( \operatorname{LimInfAvg}_2(\pi) \leq 2 \). Thus, \( E(\pi) \leq 2 \), and we establish the fact that optimal strategies may require infinite-memory strategies (in this example, the presented infinite-memory strategy is not optimal, but we demonstrated that the best finite-memory strategy does not give an optimal value).

5.4.2 Synthesis of a finite-memory controller for mean-payoff expression objectives

In this subsection we apply Theorem 5 to mean-payoff expression objectives. We first prove that the solution for mean-payoff expressions satisfies Properties 1-3, and thus the quantitative analysis problem is computable for mean-payoff expression games. We then show that the boolean analysis problem is inter-reducible with Hilbert’s tenth problem over rationals (\( H10(\mathbb{Q}) \)) by showing that an effective algorithm for \( H10(\mathbb{Q}) \) implies that mean-payoff expressions satisfy Property 4, and by a reduction from \( H10(\mathbb{Q}) \) to mean-payoff expression games.

One-player games were solved in [31] and in the previous chapter. We present our solution from Chapter 4 to establish properties of the one-player solution. For an expression \( E \) and a one-player game \( (G, v_0) \), that is, a game over graph \( G \) with initial vertex \( v_0 \), we say that a threshold \( \nu \) is feasible if the player has a strategy that achieves a value at least \( \nu \) (we recall that in the one-player setting, the player aim to maximize the value of the objective). The max-free constraints were presented in the proof of Lemma 15 of the previous chapter. They describe the feasible thresholds of a max-free expression (a threshold \( \nu \) is feasible if the one-player can achieve a value of at least \( \nu \)), and we repeat their definition for the purpose of self containment.

**Definition 1** (Max-free constraints). Let \( G \) be a strongly-connected \( k \)-dimensional game graph, and we recall that \( C(G) \) is the set of simple cycles of \( G \). Let \( E \) be a max-free expression such that the first \( j \) dimensions of \( G \) occur in \( E \) as lim-inf (and
the others as lim-sup). We define a variable $X_c^i$ for every simple cycle $c$ and index $i \in \{j + 1, \ldots, k\}$, and we define a vector of variables \( \vec{r} = (r_1, \ldots, r_{2k}) \). Then the max-free constraints for threshold $\nu \in \mathbb{Q}$ are

1. $\sum_{c \in C(G)} X_c^i \text{Avg}_m(c) \geq r_m$ for every $i \in \{j + 1, \ldots, k\}$ and $m \in \{1, \ldots, j, i\}$
2. $\sum_{c \in C(G)} X_c^i = 1$ for every $i \in \{j + 1, \ldots, k\}$
3. $X_c^i \geq 0$ for every $i \in \{j + 1, \ldots, k\}$ and $c \in C(G)$
4. $M_E \times \vec{r} \geq (0, \ldots, 0, \nu)^T$

where $M_E$ is a matrix that is independent of the graph, and computable from $E$. (We note that in the proof of Lemma 15, the first type of constraints was $\sum_{c \in C(G)} X_c^i w_m(c) \geq r_m$, where $w_m$ is the projection of $w$ to the $m$-th dimension, and the second type of constraints was $\sum_{c \in C(G)} |c| X_c^i = 1$. It is straight forward to verify that the constraints are equivalent — in terms of feasibility. In addition, the fourth constraint was presented as $M_E \times \vec{r} \geq \vec{b}_\nu$; but the proof of Lemma 15 implies that $\vec{b}_\nu = (0, \ldots, 0, \nu)^T$.) We proved in 15 that a threshold $\nu$ is feasible if and only if the corresponding max-free constraints are feasible. For a max-free expression $E$, a strongly-connected graph $G$ and a threshold $\nu$, we denote the max-free constraints by $\text{MFC}(E, G, \nu)$ and we observe that for a (normal-form) mean-payoff expression $E = \max(E_1, \ldots, E_\ell)$ and a strongly-connected graph $G$, the solution function for the one-player game is $f(G) = \max\{\nu \in \mathbb{R} | \exists i \in \{1, \ldots, \ell\} \text{ s.t } \text{MFC}(E_i, G, \nu) \text{ is feasible}\}$. By the definition of the max-free constraints, it easily follows that the solution is a function that is first-order definable and continuous (i.e., it satisfies Properties 1 and 3). In the next Lemma we prove that the solution also satisfies the second property.

**Lemma 24.** Let $E$ be a mean-payoff expression over $k$ dimensions, and let $f$ be its one-player solution function. Then for every two strongly-connected graphs $G$ and $H$: if $\text{CONVEX}(H) \subseteq \text{CONVEX}(G)$, then $f(H) \leq f(G)$.

**Proof.** Since we assume that $E = \max(E_1, \ldots, E_n)$, where $E_i$ is a max-free expression, it is enough to prove that if a threshold $\nu$ is feasible in $H$ for the max-free expression $E_i$, then it is also feasible in $G$. Let $c_1^G, \ldots, c_n^G$ and $c_1^H, \ldots, c_m^H$ be the simple cycles of $G$ and $H$ respectively. We note that since $\text{CONVEX}(H) \subseteq \text{CONVEX}(G)$, then for every convex combination $x_1, \ldots, x_m$, there is a convex combination $y_1, \ldots, y_n$ such that $\sum_{i=1}^n x_i \text{Avg}(c_i^H) = \sum_{i=1}^n y_i \text{Avg}(c_i^G)$. Hence, a solution for the max-free constraints over graph $H$ induces a solution for the max-free constraints over $G$ (by replacing, in the inequalities of constraints 1 over graph $H$, every convex combination of cycles of $H$ by the corresponding convex combination of cycles of $G$).

Thus, every threshold that is feasible for $H$ is also feasible for $G$, and the proof follows.
Hence, the one-player solution function of mean-payoff expressions satisfies Properties 1-3 and the next theorem follows.

**Theorem 6.** The quantitative analysis problem for mean-payoff expression games (where player 1 is restricted to finite-memory strategies) is computable.

We now show that the solution for one-player mean-payoff expression games satisfies Property 4 if and only if $H_{10}(\mathbb{Q})$ is decidable. We recall that $H_{10}(\mathbb{Q})$ stands for Hilbert’s tenth problem over rationals and that the corresponding problem for naturals is known to be undecidable, for reals the problem is decidable and for rationals the decidability is a long-standing open question. We first prove the direction from right to left.

**Lemma 25.** If $H_{10}(\mathbb{Q})$ is decidable, then mean-payoff expressions satisfy the fourth property.

**Proof.** Let $G$ be an arbitrary strongly connected graph with $n$ simple cycles, let $C(G) = \{C_1, \ldots, C_n\}$ be its set of simple cycles, let $E = \max(E_1, \ldots, E_m)$ be a mean-payoff expression (where $E_i$ is a max-free expression), and let $\nu$ be a rational threshold. We recall that the sentence $(\zeta_n(\text{Avg}(C_1), \ldots, \text{Avg}(C_n) \leq \nu))$ is equivalent to the statement:

For every $y > \nu$ and $i \in \{1, \ldots, m\}$, the constraints $\text{MFC}(E_i, G, y)$ are infeasible.

By the definition of the max-free constraints, when the set $\text{Avg}(G)$ is fixed the above statement is easily reduced to the infeasibility of $m$ linear systems, each of them is of the form:

$$A^i_{\text{Avg}(G)} \vec{x} \leq \vec{b}^i \quad \text{and} \quad B^i_{\text{Avg}(G)} \vec{x} < \vec{c}^i$$

By Motzkin’s Transposition Theorem (e.g., Theorem 1 in [13]) the infeasibility of a linear system $A^i_{\text{Avg}(G)} \vec{x} \leq \vec{b}^i$ and $B^i_{\text{Avg}(G)} \vec{x} < \vec{c}^i$ is equivalent to the existence of two non-negative vectors $\vec{y}, \vec{z} \geq \vec{0}$ such that either

- $\vec{z} = 0$ and $(A^i_{\text{Avg}(G)})^T \vec{y} = 0$ and $\vec{b}^iT \vec{y} < 0$; or
- $\vec{z} \neq 0$ and $(A^i_{\text{Avg}(G)})^T \vec{y} + (B^i_{\text{Avg}(G)})^T \vec{z} = 0$ and $\vec{b}^iT \vec{y} + \vec{c}^iT \vec{z} \leq 0$

Since every linear inequality has a rational solution (when the coefficients are rational) we get that if such $\vec{y}$ and $\vec{z}$ exist, then there also exist rational $\vec{y}$ and $\vec{z}$ that satisfy the above. Hence the above statement is equivalent to the rational feasibility of the following constraints (for variables $\vec{y} = (y_1, \ldots, y_r), \vec{z} = (z_1, \ldots, z_r), p_1, p_2$ and $q$):

- $p_1 > 0, p_2 > 0, q \geq 0$
- $\vec{y} \geq \vec{0}, \vec{z} \geq \vec{0}$
- $(A^i_{\text{Avg}(G)})^T \vec{y} + (B^i_{\text{Avg}(G)})^T \vec{z} = 0$
• \((\sum_{j=1}^{r} z_{j} - p_{1})(\vec{b}_{1}^{T} \vec{y} + p_{2}) = 0\)

• \((\sum_{j=1}^{r} z_{j})(\vec{b}_{1}^{T} \vec{y} + c_{1}^{T} \vec{z} + q) = 0\)

By Lagrange’s four-square Theorem, every natural number is the sum of four integer squares. Therefore, every inequality of the form \(x \geq 0\) is equivalent to the rational feasibility of the equation

\[ x = \frac{x_{1}^{2} + x_{2}^{2} + x_{3}^{2} + x_{4}^{2}}{1 + x_{5}^{2} + x_{6}^{2} + x_{7}^{2} + x_{8}^{2}} \]

and every inequality of the form \(x > 0\) is equivalent to the rational feasibility of the equation

\[ x = \frac{1 + x_{1}^{2} + x_{2}^{2} + x_{3}^{2} + x_{4}^{2}}{1 + x_{5}^{2} + x_{6}^{2} + x_{7}^{2} + x_{8}^{2}} \]

and the equations of the above form can be easily transformed into Diophantine equations. Hence, we get that the infeasibility of a linear system

\[ A_{\text{Avg}(\mathcal{G})}^{T} \vec{x} \leq \vec{b}_{1} \text{ and } B_{\text{Avg}(\mathcal{G})}^{T} \vec{x} < \vec{c}_{1}^{T} \]

is equivalent to the rational feasibility of several Diophantine equations \(D_{1} = 0, D_{2} = 0, \ldots, D_{r} = 0\), and therefore it is equivalent to the rational feasibility of \(D^{i} = D_{1}^{2} + \cdots + D_{r}^{2} = 0\). Therefore, when the set of simple cycles is fixed, the simultaneous infeasibility of all the max-free constraints is equivalent to the rational feasibility of the Diophantine equation \(D_{\text{Avg}(\mathcal{G})} = \sum_{i=1}^{m} (D_{i})^{2} = 0\). We also note that if \(\text{Avg}(\mathcal{G})\) is not fixed, that is \(\text{Avg}(\mathcal{C}_{i})\) is a vector of variables (for \(i = 1, \ldots, n\)), then \(D_{\text{Avg}(\mathcal{G})} = 0\) remains a Diophantine equation.

We are now ready to prove that the solution for one-player mean-payoff games is satisfies Property 4 (if \(H10(\mathbb{Q})\) is decidable). For \(n\) matrices \(A_{1}, \ldots, A_{n}\) the rational satisfaction of \(\zeta_{n}(A_{1}x_{1}, \ldots, A_{n}x_{n}) \leq \nu\) is equivalent to the existence of a rational solution to \(D_{\text{Avg}(\mathcal{G})} = 0\) (for \(\text{Avg}(\mathcal{G}) = \{A_{1}x_{1}, \ldots, A_{n}x_{n}\}\)). We can encode the requirement that \(x_{1}, \ldots, x_{n} \in \text{QST}\) by a Diophantine equations \(K(x_{1}, \ldots, x_{n}) = 0\) by the same techniques we used for the construction of \(D_{\text{Avg}(\mathcal{G})} = 0\). Hence, the satisfaction of \(\zeta_{n}(A_{1}x_{1}, \ldots, A_{n}x_{n}) \leq \nu\) is equivalent to the existence of a rational solution to the Diophantine equation \(K^{2} + D^{2} = 0\), and if \(H10(\mathbb{Q})\) is decidable, then we can effectively determine whether \(K^{2} + D^{2} = 0\) has a rational solution and the proof follows.

We now prove the reduction from \(H10(\mathbb{Q})\) to the boolean analysis of mean-payoff expression games, and we show that there is a reduction even for a simpler subclass of mean-payoff expressions. An expression \(E\) is sum-free and LimInfAvg-only if only the min and max operators occur in \(E\) and all the atomic expressions in \(E\) are of the form LimInfAvg\(_{i}\). (In addition, the numerical complement operator also does not occur). The next lemma shows that the boolean analysis problem for sum-free LimInfAvg-only
expressions is $H10(\mathbb{Q})$-hard, and we postpone the proof of the lemma to the next section.

**Lemma 26.** If the boolean analysis problem is decidable for sum-free $\text{LimInfAvg}$-only expressions, then $H10(\mathbb{Q})$ is decidable.

The next theorem summarizes the results of Lemmas 26 and 25.

**Theorem 7.** The boolean analysis problem for mean-payoff expression games (when player 1 is restricted to finite-memory strategies) is inter-reducible with $H10(\mathbb{Q})$, and it is $H10(\mathbb{Q})$-hard even for sum-free $\text{LimInfAvg}$-only expressions.

We also consider the case where both players are restricted to finite-memory strategies. In this setting, the quantitative analysis problem is to compute $\inf_{\sigma \in \mathcal{F}, M_1} \sup_{\tau \in \mathcal{F}, M_2} \text{Val}_{\sigma, \tau}$. The boolean analysis problem is to determine whether player 1 has a finite-memory strategy that assures a value of at most $\nu$ against any player-2 finite-memory strategy.

**Theorem 8.** When both players are restricted to finite-memory strategies: (i) the quantitative analysis problem for mean-payoff expression games is computable; (ii) the boolean analysis problem for mean-payoff expression games is inter-reducible with $H10(\mathbb{Q})$, and it is $H10(\mathbb{Q})$-hard even for sum-free $\text{LimInfAvg}$-only expressions.

**Proof.** When both players are restricted to finite-memory strategies the outcome of the game is an ultimately periodic path $\pi = \pi_1(\pi_2)^\omega$. Thus, for every dimension $i$ we have $\text{LimInfAvg}_i(\pi) = \text{LimSupAvg}_i(\pi)$. Hence, w.l.o.g we may assume that the game objective is a $\text{LimInfAvg}$-only expression. In order to prove the theorem, we will show a reduction from games in which both players are restricted to finite-memory strategies to games in which only player 1 is restricted to finite-memory strategies. The reduction is based on the next lemma.

**Lemma 27.** Let $E$ be a $\text{LimInfAvg}$-only expression and let $G$ be a multidimensional weighted graph, and the goal of player 1 is to assure $E \leq \nu$. Then a player-1 finite-memory strategy is winning if and only if it wins against every player-2 finite-memory strategy.

**Proof.** The proof for the direction from left to right is trivial. To prove the converse direction we fix a player-1 finite-memory strategy $\sigma$ and we show that if player 2 has strategy that wins against $\sigma$, then he also has a finite-memory winning strategy that wins against $\sigma$. We note that when $\sigma$ is fixed, a player-2 strategy is an infinite path in $G' \sigma$ and a player-2 finite-memory strategy is an ultimately periodic path in $G' \sigma$. Hence, there exists an infinite path $\pi$ in $G' \sigma$ for which $E$ assigns a value greater than $\nu$. We claim that for every $\epsilon > 0$ there is an ultimately periodic path $\rho$ in $G' \sigma$ such that in every dimension $\text{LimInfAvg}_i(\rho) \geq \text{LimInfAvg}_i(\pi) - \epsilon$. Indeed, let $s$ be a state that is visited infinitely often by $\pi$, and let $\pi_s$ be a suffix of $\pi$ that begins in state
CHAPTER 5. FINITE-MEMORY STRATEGY SYNTHESIS FOR 
MULTIDIMENSIONAL MEAN-PAYOFF OBJECTIVES

\( s \), and we observe that \( \text{LimInfAvg}(\pi_s) = \text{LimInfAvg}(\pi) \). By the definition of \( \text{LimInfAvg} \) and by the finiteness of the graph it follows that for every \( \epsilon > 0 \) there exists a path \( \pi_\epsilon \) that is a prefix of \( \pi_s \), ends in state \( s \), and \( \text{LimInfAvg}(\pi_\epsilon) \geq \text{LimInfAvg}(\pi_s) - \epsilon \).

We denote by \( \pi_0 \) the shortest path from the initial state to \( s \), and we get that the ultimately periodic path \( \rho_\epsilon = \pi_0(\pi_\epsilon)^\omega \) satisfies the assertion of the claim. To complete the proof of the lemma, we denote the number of sum operators in \( E \) by \# sum and we set \( \epsilon = \frac{E(\pi) - \nu}{2 \# \text{sum}} \). It is easy to verify that the ultimately periodic path \( \rho_\epsilon \) satisfies \( E(\rho_\epsilon) \geq E(\pi) - \frac{E(\pi) - \nu}{2} = \frac{E(\pi) + \nu}{2} > \nu \), and the proof follows.

The proof of Theorem 8 follows immediately from the fact that we only consider \( \text{LimInfAvg} \)-only expressions and from Lemma 27 and Theorems 6 and 7.

As a final remark, we note that while the boolean analysis for sum-free \( \text{LimInfAvg} \)-only expressions is \( H10(Q) \)-hard when player 1 is restricted to a finite-memory strategy (and also when both players are restricted to finite-memory strategies), the next lemma shows that the problem is decidable when both players may use arbitrary strategies.

**Lemma 28 (Theorem 5 in [99]).** When both players may use arbitrary strategies, the boolean analysis of sum-free \( \text{LimInfAvg} \)-only expression games is decidable.

**Proof.** The proof follows from Theorem 5 in [99] due to the fact that there is an immediate translation from sum-free \( \text{LimInfAvg} \)-only expressions to the \( \bigvee \bigwedge \text{MeanPayoffInf}^\leq(\nu) \) objectives that were defined in [99].

5.5 Proof of Lemma 26

We prove Lemma 26 in the next three subsections. In the first subsection we present an alternative formulation for \( H10(Q) \). In the second subsection we prove a simple technical lemma on vectors. In the third subsection we present a reduction from the problem we presented in the first subsection to sum-free \( \text{LimInfAvg} \)-only games, and the reduction relies on the lemma that we prove in the second subsection.

We note that the first two subsection are technical and tedious, but they relay only on basic algebra.

5.5.1 Alternative formulations of \( H10(Q) \)

In this subsection, we present five problems; the first problem is \( H10(Q) \), and we show a reduction from the i-th problem to the \( i + 1 \)-th problem, for \( i = 1, 2, 3, 4 \). Thus, we get that there is a reduction from \( H10(Q) \) to the fifth problem (that is, Problem 6), and in the third subsection we will show a reduction from that problem to mean-payoff expression games.
CHAPTER 5. FINITE-MEMORY STRATEGY SYNTHESIS FOR
MULTIDIMENSIONAL MEAN-PAYOFF OBJECTIVES 65

Problem 2 \((H10(\mathbb{Q}))\). For a polynomial \(P\), find a rational solution to

\[ P(q_1, \ldots, q_n) = 0 \]

Problem 3. Find a rational solution to

\[ q_0 \cdot P\left(\frac{q_1}{q_0}, \ldots, \frac{q_n}{q_0}\right) = 0 \]

(for a polynomial \(P\)) subject to

- \(q_0 \leq q_i\) for every \(i = 1, \ldots, n\); and
- \(q_i \geq 1\) for every \(i = 0, \ldots, n\).

Lemma 29. There is a reduction from \(H10(\mathbb{Q})\) to Problem 3.

Proof. We first note that we can easily reduce \(H10(\mathbb{Q})\) to the problem of finding a rational solution for the polynomial equation \(D(q_1, q_2, \ldots, q_n) = 0\) subject to \(q_1, q_2, \ldots, q_n \geq 1\). (The reduction is trivial, a polynomial equation \(P(q_1, \ldots, q_n) = 0\) has a solution if and only if the polynomial equation \(D(p_1, \ldots, p_{2n}) = P(p_1 - p_2, p_3 - p_4, \ldots, p_{2n-1} - p_{2n}) = 0\) has a solution that satisfies \(p_i \geq 1\).) We define \(P = D\) and we note that \(q_0 \cdot P = 0\) has a rational solution (subject to \(q_0 \geq 1\)) if and only if \(P = 0\) has a rational solution, and it is trivial to observe that \(P = 0\) has a rational solution (subject to \(q_0 \geq q_i\) and \(q_i \geq 1\)) if and only if \(D = 0\) has a rational solution (subject to \(q_i \geq 1\)).

\(\square\)

Problem 4. For a given a set of variables \(Q = \{q_1, \ldots, q_n\}\), and a set of equations such that at most one equation is of the form

\[ \sum_{i \in I} \alpha_i q_i = 0, \text{ for some } I \subseteq \{0, \ldots, n\} \text{ and } \alpha_i \in \mathbb{Q} \text{ for every } i \in I. \]

and all the other equations are of the form

\[ q_iq_j = q_kq_\ell \text{ for some } i, j, k, \ell \in \{0, \ldots, n\}. \]

find a rational solution that satisfies \(1 \leq q_0 \leq q_i\) for every \(i = 1, \ldots, n\).

Lemma 30. There is a reduction from Problem 3 to Problem 4.

Proof. We prove the lemma by giving a generic example that demonstrates the reduction. Suppose that the equation with the form of Problem 3 is \(q_0 \cdot P\left(\frac{q_1}{q_0}, \ldots, \frac{q_n}{q_0}\right) = 5q_1^2q_2q_3^3 + \frac{q_1^2}{q_0} + 7q_0\), then we reduce it to a problem with the form of Problem 4 by defining the following equations:

- \(p_0 \cdot q_0 = q_0 \cdot q_0\) (equivalent to \(p_0 = q_0\))
- \(p_1 \cdot q_0 = q_1 \cdot q_1\) (equivalent to \(p_1 = \frac{q_1}{q_0}\))
• \( p_2 \cdot q_0 = q_3 \cdot q_3 \) and \( p_3 \cdot q_0 = p_2 \cdot q_3 \) (equivalent to \( p_3 = \frac{q_3^2}{q_0} \))

• \( p_4 \cdot q_0 = p_1 \cdot p_3 \) (equivalent to \( p_4 = \frac{q_4^2}{q_0} \))

• \( p_5 \cdot q_0 = p_4 q_2 \) (equivalent to \( p_5 = \frac{q_5^2 q_2}{q_0} \))

5p_5 + p_1 + 7q_0 = 0, subject to 1 ≤ q_0 ≤ q_1, q_2, q_3, p_0, p_1, p_2, p_3, p_4, p_5 and \( q_i, p_j ≥ 1 \) (equivalent to \( q_0 \cdot P(\frac{q_0}{q_0}, \ldots, \frac{q_0}{q_0}) = 0 \))

A solution to the above equations that satisfies 1 ≤ q_0 ≤ q_1, q_2, q_3, p_0, p_1, p_2, p_3, p_4, p_5 is clearly a solution for \( q_0 \cdot P = 0 \) that satisfies Problem 3 conditions. Conversely, a solution to \( q_0 \cdot P = 0 \) that satisfies Problem 3 conditions is a solution for the above constraints, and since 1 ≤ q_0 ≤ q_1, q_2, q_3 we also get that \( q_0 ≤ p_1, p_2, p_3, p_4, p_5 \) and a solution to the above equations follows.

Problem 5. For a given sets of variables \( Q = \{q_1, \ldots, q_n\} \), \( P = \{p_1, \ldots, p_n\} \), and a given set of equations, each of the form of either:

• \( \sum_{i=1}^{I} \alpha_i q_i = 0 \), for some \( I \subseteq \{1, \ldots, n\} \); or

• \( q_ip_j = q_k p_{\ell} \), for some \( i, j, k, \ell \in \{1, \ldots, n\} \); or

• \( q_i = \frac{1}{2} \sum_{j=1}^{n} q_j \), for some \( i \in \{1, \ldots, n\} \); or

• \( p_i = \frac{1}{2} \sum_{j=1}^{n} p_j \), for some \( i \in \{1, \ldots, n\} \); or

find a rational solution that satisfies

• \( q_i ≤ q_i \), for \( i = 1, \ldots, n \); and

• \( q_i, p_i ≥ 1 \) for \( i = 1, \ldots, n \); and

• \( \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} q_i \).

Lemma 31. There is a reduction from Problem 4 to Problem 5.

Proof. To show a reduction, we need to show how to encode an equation of the form of \( q_1 q_2 = q_3 q_4 \) with equations of the above form. For this purpose we define the equations:

• \( q_{n+1} = \frac{1}{2} \sum_{j=1}^{n+1} q_j \) and \( p_{n+1} = \frac{1}{2} \sum_{j=1}^{n+1} p_j \)

• \( q_2 p_{n+1} = q_{n+1} p_1 \)

• \( q_4 p_{n+1} = q_{n+1} p_2 \)

• \( q_1 p_1 = q_3 p_2 \)

It is straight forward to observe that if \( \sum_{j=1}^{n+1} q_j = \sum_{j=1}^{n+1} p_j \) then the above set of equations are equivalent to \( q_1 q_2 = q_3 q_4 \).
Problem 6. For a given sets of variables $Q = \{q_1, \ldots, q_n\}$, $P = \{p_1, \ldots, p_n\}$, and a given set of constraints, each of the form of either:

- $\sum_{i \in I} \alpha_i q_i \leq 0$, for some $I \subseteq \{1, \ldots, n\}$; or
- $\sum_{i \in I} \alpha_i p_i \leq 0$, for some $I \subseteq \{1, \ldots, n\}$; or
- $q_i p_j = q_k p_\ell$ for some $i, j, k, \ell \in \{1, \ldots, n\}$

find a rational solution that satisfies

- $q_i, p_i > 0$ for $i = 1, \ldots, n$

**Lemma 32.** There is a reduction from Problem 5 to Problem 6.

**Proof.** The reduction is straightforward. We replace every equation of the form of $\sum_{i \in I} \alpha_i q_i = 0$ with two constraints $\sum_{i \in I} \alpha_i q_i \leq 0$ and $\sum_{i \in I} -\alpha_i q_i \leq 0$. We replace $q_i = \frac{1}{2} \sum_{j=1}^n q_j$ with $\sum_{j \in \{1, \ldots, n\} - \{i\}} \frac{1}{2} q_j - \frac{1}{2} q_i \leq 0$ and $\sum_{j \in \{1, \ldots, n\} - \{i\}} -\frac{1}{2} q_j + \frac{1}{2} q_i \leq 0$. We replace $p_i = \frac{1}{2} \sum_{j=1}^n p_j$ with $\sum_{j \in \{1, \ldots, n\} - \{i\}} \frac{1}{2} p_j - \frac{1}{2} p_i \leq 0$ and $\sum_{j \in \{1, \ldots, n\} - \{i\}} -\frac{1}{2} p_j + \frac{1}{2} p_i \leq 0$. In addition, we add $n$ constraints $q_i \leq q_i$ for $i = 1, \ldots, n$. It is straightforward to observe that if the above formed constraints have a rational solution $Q = \{q_1, \ldots, q_n\}, P = \{p_1, \ldots, p_n\}$ that satisfies $q_i, p_i > 0$, then for every rational positive $m$ we get that $mQ = \{mq_1, \ldots, mq_n\}, P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}, mp = \{mp_1, \ldots, mp_n\}$ are also solutions. Hence, a solution to the formed constraints implies that there is a solution that satisfies $q_i, p_i \geq 1$ and $\sum_{i=1}^n p_i = \sum_{i=1}^n q_i$. And conversely, if the formed constraints are not satisfiable, then clearly the original equations are not solvable.

\[\square\]

### 5.5.2 Auxiliary lemma

In this subsection, we prove the next lemma.

**Lemma 33.** Let $\alpha_1, \alpha_2, \beta_1, \beta_2$ be strictly positive rationals, and let $v_1(\alpha_1) = \alpha_1 \cdot (-1, 0, 1, 0)$, $v_2(\alpha_2) = \alpha_2 \cdot (0, 1, 0, -1)$, $u_1(\beta_1) = \beta_1 \cdot (1, 0, -1, 0)$, and $u_2(\beta_2) = \beta_2 \cdot (0, -1, 0, 1)$. For every $m, n \in \mathbb{Q}$ we denote by the vector $x(m, n) = (x_1, x_2, x_3, x_4)$ the sum $m(v_1 + v_2) + n(u_1 + u_2)$. Then the following assertions are equivalent:

1. $\frac{v_1}{\alpha_1} = \frac{\beta_1}{\alpha_1}$

2. For every non-negative rationals $m, n$: $\max(\min(x_1, x_2), \min(x_3, x_4)) \leq 0$.

**Proof.** By definition $x_1 = -ma_1 + n\beta_1$, $x_2 = ma_2 - n\beta_2$, $x_3 = -x_1$ and $x_4 = -x_2$.

We first prove that assertion 1 implies assertion 2. Suppose that $\frac{v_1}{\alpha_1} = \frac{\beta_1}{\alpha_1}$, let $m$ and $n$ be arbitrary non-negative rationals, and we denote $k = \frac{m}{n}$. In order to prove that $\max(\min(x_1, x_2), \min(x_3, x_4)) \leq 0$, it is enough to show that if $x_1 > 0$, then $x_2 < 0$ (since in this case $x_3 = -x_1 < 0$). Suppose that $x_1 > 0$. Hence, $\beta_1 > k\alpha_1$, and we get that $k < \frac{\alpha_1}{\beta_1}$. By definition, $x_2 = n(k\alpha_2 - \beta_2)$, and since we assumed that $\frac{\alpha_1}{\beta_1} = \frac{\beta_1}{\alpha_1}$,
and we proved that \( k < \frac{\beta_1}{\alpha_1} \), we get that \( x_2 < 0 \), and the claim that assertion 1 implies assertion 2 follows.

In order to prove that assertion 2 implies assertion 1, we consider two distinct cases. In the first case we assume (towards a contradiction) that \( \frac{\beta_1}{\alpha_1} > \frac{\beta_2}{\alpha_2} \) and we choose \( m \) and \( n \) that satisfy \( \frac{\beta_1}{\alpha_1} > k = \frac{m}{n} > \frac{\beta_2}{\alpha_2} \). We claim that \( x_1 > 0 \) and \( x_2 > 0 \), and therefore a contradiction to the assumption that \( \max(\min(x_1,x_2),\min(x_3,x_4)) \leq 0 \) follows. Indeed, since \( \frac{\beta_1}{\alpha_1} > k \), then \( x_1 = n(-k\alpha_1 + \beta) > 0 \), and since \( k > \frac{\beta_2}{\alpha_2} \), then \( x_2 = n(k\alpha_2 - \beta_2) > 0 \). In the second case, we assume that \( \frac{\beta_1}{\alpha_1} < \frac{\beta_2}{\alpha_2} \) and by similar arguments, we get that \( x_3, x_4 > 0 \) and a contradiction follows. Hence, in both cases we get that assertion 2 implies assertion 1, and the proof of the lemma follows.

\[ \square \]

### 5.5.3 The reduction

In this subsection, we present a reduction from Problem 6 to the boolean synthesis problem for mean-payoff expressions (when player 1 is restricted to finite-memory strategies). The reduction is as following: For a given sets of variables \( Q = \{q_1, \ldots, q_n\} \), \( P = \{p_1, \ldots, p_n\} \), and a given set of constraints, each of the form of either:

- \( \sum_{i \in I} \alpha_i q_i \leq 0 \), for some \( I \subseteq \{1, \ldots, n\} \); or

- \( \sum_{i \in I} \alpha_i p_i \leq 0 \), for some \( I \subseteq \{1, \ldots, n\} \); or

- \( q_i p_j = q_k p_\ell \) for some \( i, j, k, \ell \in \{1, \ldots, n\} \)

We denote by \( t_1 \) the number of constraints that are of the first form, and w.l.o.g we assume that the number of constraints that are of the second form is also \( t_1 \). We denote by \( t_2 \) the number of constraints that are of the third form. We construct a \( k = 2 + n + t_1 + 4t_2 \) dimensional game graph with 5 states (see Figure 5.8), and an expression \( E = \max(\text{LimInfAvg}_1, \ldots, \text{LimInfAvg}_{2+n+t_1}, E_1, \ldots, E_{2t_2}) \) where

\[ E_i = \min(\text{LimInfAvg}_{2+n+t_1+2i}, \text{LimInfAvg}_{2+n+t_1+2i+1}) \]

The transitions of the graph are described in Figure 5.8, and each of the states \( a_2 \) and \( b_2 \) has \( n \) self-loop edges.
The weight vector $\vec{w}$ of the $i$-th self-loop edge of state $a_2$ is determined according to the next rules:

1. The first two dimensions of $\vec{w}$ are $-1$ and $+1$ (respectively). Intuitively, this assures that player 1 will not stay forever in state $a_1$ or in state $a_2$.

2. The weight of dimension $2 + i$ is $-1$ and for $j \in \{1, \ldots, n\} - \{i\}$ the weight of dimensions $j$ is 0. Intuitively, this assures that player 1 will visit edge $i$ at least once.

3. If the $j$-th type-1 equation is $\sum_{m \in I} \alpha_m q_m \leq 0$, then if $i \in I$, then the weight in dimension $2 + n + j$ is $-\alpha_m$. Otherwise, we assign zero for this dimension. Intuitively, this enforces player 1 to visits edge $i$ for $q_i$ times in such way that $\sum_{m \in I} \alpha_m q_m \leq 0$.

4. If the $j$-th type-3 equation is $q_m p_r = q_k p_\ell$, then the weights of the four dimensions $2 + n + t_1 + 4j, 2 + n + t_1 + 4j + 1, 2 + n + t_1 + 4j + 2, 2 + n + t_1 + 4j + 3$ are:
   - If $i = m$, then the weights are $(-1, 0, 1, 0)$
   - If $i = k$, then the weights are $(0, 1, 0, -1)$
   - Otherwise, the weights are $(0, 0, 0, 0)$

The weight vector $\vec{w}$ of the $i$-th self-loop edge of state $b_2$ is determined according to the next rules:
1. The first $2 + n$ dimensions are determined by the same rules that we presented to the self-loop edges of state $a_2$.

2. If the $j$-th type-2 equation is $\sum_{m\in I} \alpha_m p_m \leq 0$, then if $i \in I$, then the weight in dimension $2 + n + j$ is $-\alpha_m$. Otherwise, we assign zero for this dimension. Intuitively, this enforces player 1 to visit edge $i$ for $p_i$ times in such a way that $\sum_{m\in I} \alpha_m p_m \leq 0$.

3. If the $j$-th type-3 equation is $q_m p_r = q_k p_\ell$, then the weights of the four dimensions $2 + n + t_1 + 4j, 2 + n + t_1 + 4j + 1, 2 + n + t_1 + 4j + 2, 2 + n + t_1 + 4j + 3$ are:
   - If $i = m$, then the weights are $(1, 0, -1, 0)$
   - If $i = k$, then the weights are $(0, -1, 0, 1)$
   - Otherwise, the weights are $(0, 0, 0, 0)$

In the rest of this subsection, we will prove that player 1 has a finite-memory strategy that assures non-positive value for the expression $E$ if and only if the given set of equations has a solution that satisfies Problem 6 conditions.

In the next lemmas we prove key properties of the game. The first lemma characterized the one-player game solution for the expression $E$.

**Lemma 34.** Let $G$ be an arbitrary strongly connected $k$-dimensional weighted one-player game graph, and let $f$ be the one-player solution for the expression $E$. Then $f(G) > 0$ if and only if
   - $G$ has a simple cycle with positive average weight in a dimension $i \in \{1, \ldots, 2 + n + t_1\}$; or
   - $G$ has two simple cycles $C_1$ and $C_2$, and there exist an index $i \in \{1, \ldots, 2t_2\}$ and two positive rationals $m, n$ for which
     $$m\text{Avg}(C_1) + n\text{Avg}(C_2)$$
     is positive is dimension $2 + n + t_1 + 2i$ and in dimension $2 + n + t_1 + 2i + 1$.

**Proof.** The proof follows directly by the definitions of the max-free constraints (Definition 1).

**Lemma 35.** In the mean-payoff expression game over game graph $G$ (that is constructed by the reduction) and threshold 0, player 1 wins from vertex $s_0$ if and only if he has a finite-memory strategy $\sigma$ such that $f(\text{CONVEX}(G^\sigma)) \leq 0$ (where $f$ is the one-player solution for the expression $E$).

**Proof.** By the construction of $G$ it follows that if player 1 strategy is to loop for ever in state $a_1$ or $b_1$, then the lim-inf of the average weight in dimension 1 will be 1 and $E$ will get a positive value. Similarly, if player 1 strategy is to loop forever in state $b_2$ or $a_2$,
then the average weight in dimension 2 is positive, and so does the value of $E$. Hence, every player-1 winning strategy will visit the initial state $s_0$ infinitely often. Therefore, if $\sigma'$ is a player-1 winning strategy, then every SCC in $G^{\sigma'}$ contains a vertex $(s_0, m)$ (for some memory state $m$). Let $S$ be a terminal SCC in $G^{\sigma'}$ and let $(s_0, m)$ be a vertex in $S$. We construct the witness strategy $\sigma$ by changing the initial memory state of $\sigma'$ to $m$. If $\sigma'$ is a winning strategy, then by definition $f(S) \leq 0$ and since $G^\sigma = S$ we get that $f(\text{CONVEX}(G)) \leq 0$.

Hence, if player 1 wins in the game, then such $\sigma$ exists, and the proof for the converse direction is trivial (since such a strategy $\sigma$ is a winning strategy).

\[ \Box \]

In the game graph $G$ player 2 has only two possible memoryless strategies: the first strategy is to follow the edge $(s_0, a_1)$, and we denote this strategy by $\tau_1$, and the second strategy is to follow $(s_0, b_1)$, and we denote it by $\tau_2$.

**Lemma 36.** There exists a player-1 strategy for which $f(G^\sigma) \leq 0$ if and only if there exist cyclic paths $\pi_1$ and $\pi_2$ such that $\pi_1$ is a cyclic path in $G^{\tau_1}$ that visits all the edges of $G^{\tau_1}$ and $f(\text{CONVEX}(\pi_1, \pi_2)) \leq 0$.

**Proof.** By Lemma 19 such $\sigma$ exists if and only if there exist two ultimately periodic paths $\rho_1$ and $\rho_2$ such that $\rho_i$ is an infinite path in the graph $G^{\tau_i}$ and $f(\text{CONVEX}(\text{Avg}(\rho_1), \text{Avg}(\rho_2))) \leq 0$. Hence, the proof for the direction from right to left follows. In order to prove the converse direction we assume that such $\rho_1$ and $\rho_2$ exist and show how to construct $\pi_1$ and $\pi_2$. Let $\rho_1 = \pi_0(\pi_1)^\omega$ (i.e., $\pi_1$ is the periodic finite path in $\rho_1$). We claim that if $\pi_1$ does not contain all the edges of $G^{\tau_1}$, then $f(\{\text{Avg}(\pi_1)\}) > 0$. The proof of the claim is by considering the following distinct cases:

- **Case 1:** if $\pi_1$ contains only the cycles $s_0 \rightarrow a_1 \rightarrow a_2 \rightarrow s_0$, then the value of $\text{Avg}(\pi)$ is positive in the third dimension.
- **Case 2:** if $\pi_1$ contains only the self loop of $a_1$, then the value of the first dimension of $\text{Avg}(\pi)$ is positive.
- **Case 3:** if $\pi_1$ does not contain the self loop of $a_1$, and contains some of the self loops of $a_2$, then the second dimension of $\text{Avg}(\pi)$ is positive.
- **Case 4:** if $\pi_1$ contains the cycle $s_0 \rightarrow a_1 \rightarrow a_2 \rightarrow s_0$, the self loop of $a_1$ and not the $i$-th self loop of $a_2$, then dimension $2 + i$ of $\text{Avg}(\pi)$ is positive.

Hence, if $\pi_1$ does not contain all the edges of $G^{\tau_1}$, then we get that $f(\text{Avg}(\rho_1)) > 0$ (since $\text{Avg}(\rho_1) = \text{Avg}(\pi_1)$), and since $f$ is monotone, we get that $f(\text{CONVEX}(\text{Avg}(\rho_1), \text{Avg}(\rho_2))) > f(\text{Avg}(\rho_1)) > 0$, which contradict the definition of $\rho_1$. We construct the witness path $\pi_2$ in a similar way (i.e., by defining $\rho_2 = \pi_0(\pi_2)^\omega$, and the proof that $\pi_2$ contains all the edges of $G^{\tau_2}$ is similar. Since $\text{Avg}(\pi_1) = \text{Avg}(\rho_1)$, we get that $f(\text{CONVEX}(\text{Avg}(\pi_1), \text{Avg}(\pi_2))) \leq 0$ and the proof is complete.

\[ \Box \]
We now give two additional definitions and then prove the correctness of the reduction. Let $C_1,\ldots,C_n$ be the simple cycles of $G^v$. We denote $QSI(G^v) = \{ v \in \mathbb{Q}^k \mid \exists(x_1,\ldots,x_n) \in QSI(n) \text{ s.t. } v = \sum_{i=1}^n x_i \text{Avg}(C_i) \}$, and we similarly define $QSI(G^w)$. We say that two vectors $v_1$ and $v_2$ are satisfactory if $f(\text{CONVEX}(v_1, v_2)) \leq 0$. We are now ready to prove the correctness of the reduction, and by Lemma 35 and Lemma 36 it is enough to prove that there exists $v_i \in QSI(G^v)$ (for $i = 1, 2$) such that $v_1, v_2$ are satisfactory vectors if and only if the given set of equations has a rational solution.

We first prove the direction from right to left. Suppose that the given set of equations has a rational solution $P, Q$ that satisfies $q_i, p_i > 0$. We construct the vector $v_1 \in QSI(G^v)$ by taking $\frac{1}{2\sum_{i=1}^n q_i}$ fraction of the average weight of the cycle $s_0 \rightarrow a_1 \rightarrow a_2 \rightarrow s_0$, $\frac{2\sum_{i=1}^n q_i}{1+2\sum_{i=1}^n q_i}$ fraction of the average weight of the self loop of $a_1$ and $\frac{q_i}{2\sum_{i=1}^n q_i}$ fraction of the average weight of the $i$-th self loop of $a_2$. Similarly, we construct the vector $v_2 \in QSI(G^w)$ by taking $\frac{1}{1+2\sum_{i=1}^n p_i}$ fraction of the average weight of the cycle $s_0 \rightarrow b_1 \rightarrow b_2 \rightarrow s_0$, $\frac{2\sum_{i=1}^n p_i}{1+2\sum_{i=1}^n p_i}$ fraction of the average weight of the self loop of $b_1$ and $\frac{p_i}{1+2\sum_{i=1}^n p_i}$ fraction of the average weight of the $i$-th self loop of $b_2$. By the construction of $G$, and since $P$ and $Q$ are solutions for the equations, it is straightforward to verify that the first $2 + n + t_1$ dimensions of $v_1$ and $v_2$ are non-positive. In addition, by Lemma 33, and since $P$ and $Q$ satisfies all the equations of the form $q_ip_j = q_ik_i$, we get that for every positive $m, n \in \mathbb{Q}$ we have that $mv_1 + nv_2$ are non-positive in dimension $2 + n + t_1 + 2i$ or in dimension $2 + n + t_1 + 2i + 1$ for every $i = 1,\ldots,2t_4$. Hence, by Lemma 34, the vectors $v_1, v_2$ are satisfactory.

Conversely, suppose that there exist $v_i \in QSI(G^v)$ (for $i = 1, 2$) such that $v_1, v_2$ are satisfactory vectors. We denote by $w_{s_0,a}$ the average weight of the cycle $s_0 \rightarrow a_1 \rightarrow a_2 \rightarrow s_0$, by $w_{a_1}$ the average weight of the self loop of $a_1$, and by $w_{b_2}^i$ the average weight of the $i$-th self loop of $a_2$. By definition, there exists $n + 2$ positive rationals $x, y, q_1,\ldots,q_n$ for which $v_1 = xw_{s_0,a} + yw_{a_1} + \sum_{i=1}^n q_iw_{b_2}^i$. Similarly, we denote by $w_{s_0,b}$ the average weight of the cycle $s_0 \rightarrow b_1 \rightarrow b_2 \rightarrow s_0$, by $w_{b_1}$ the average weight of the self loop of $b_1$, and by $w_{b_2}^i$ the average weight of the $i$-th self loop of $b_2$, and by definition, there exists $n + 2$ positive rationals $x, y, p_1,\ldots,p_n$ for which $v_1 = xw_{s_0,b} + yw_{b_1} + \sum_{i=1}^n p_iw_{b_2}^i$. We claim the $Q = \{q_1,\ldots,q_n\}, P = \{p_1,\ldots,p_n\}$ are a solution to the given set of equations. By Lemma 34 and by the construction of the graph, it immediately follows that $Q$ and $P$ satisfy all the type-1 and type-2 constraints. In addition, by Lemma 33 (and by Lemma 34) we get that all the type-3 equations are also satisfied. Hence, we get that if there exist $v_i \in QSI(G^v)$ (for $i = 1, 2$) such that $v_1, v_2$ are satisfactory vectors, then the given set of constraints have a solution.

To conclude, we get that the boolean analysis problem for mean-payoff expressions is harder than $H10(\mathbb{Q})$, and the proof of Lemma 26 follows.

5.6 Discussion

In this work we studied the synthesis of finite-memory strategies for games with robust multidimensional mean-payoff objectives, and we obtained two main results. The first
is a positive result, namely, the computability of the quantitative analysis problem. The second has a negative flavour, and it shows that the boolean analysis is inter-reducible with Hilbert’s Tenth problem over rationals. From a practical point of view, the positive result is the most interesting, since for the first time (to the best of our knowledge) a recipe is given for computing $\epsilon$-optimal finite-memory strategies for a robust class of quantitative objectives. A future work is to investigate whether the construction of these $\epsilon$-optimal strategies is feasible, both in terms of memory size and computational complexity. From the theoretical point of view, the negative result is a bit surprising since it suggests that the boolean analysis is harder than the optimization problem, and in computer science typically there is a naive reduction from optimization problems to the corresponding decision problems. However, in our case, the optimization computes only the greatest upper bound, and since optimal finite-memory strategies need not exist, then the reduction fails. In fact, the hardness result suggests that it is even $H10(\mathbb{Q})$-hard to determine whether an optimal strategy exists.
Chapter 6

Multidimensional Mean-Payoff Games are Undecidable

In this chapter we prove that games with general multidimensional mean-payoff objectives and arbitrary (infinite-memory) strategies are undecidable. In most of this chapter, for simplicity, we consider boolean multidimensional objectives, that is, boolean formulas over $x_i \sim \nu$, where $x \in \{\text{LimInfAvg, LimSupAvg}\}$ and $\sim \in \{\geq, >, <, \leq\}$. In the last section we extend the proof for mean-payoff expressions.

Related work. All previous related works consider only restricted special cases of two-player games over multidimensional mean-payoff games. The restrictions are in three orthogonal directions, namely, restriction to one-player games, restriction to less general winning conditions and restriction to finite-memory strategies. We list the previous works below:

- One-player games. The model checking problem (one-player game) for multidimensional mean-payoff objectives (with some extensions) was considered in Chapter 4 and in [7, 21, 31, 93, 96] and decidability was established.

- Restricted winning conditions. Two-player games for restricted subclasses that contain only conjunction of atoms (e.g., $\text{LimInfAvg}_1 \geq 0 \land \text{LimInfAvg}_2 < 0 \land \text{LimSupAvg}_3 > 0$) were studied in [20, 37, 39, 98] (and in Chapter 3) and tight complexity bounds were obtained (and in particular, the problem was proved to be decidable). In [99] a subclass that contains disjunction and conjunction of atoms of the form $\text{LimInfAvg}_i \sim \nu_i$ for $\sim \in \{\geq, >\}$ (e.g., $\text{LimInfAvg}_1 \geq 0 \land \text{LimInfAvg}_2 > 0 \lor \text{LimInfAvg}_3 \geq 0$) was studied and decidability was shown.

- Finite-memory strategies. In [97] (and in the previous chapter) we considered the same objectives but restricted player-1 to play only with finite-memory strategies. We showed that the problem is provably hard to solve and its decidability is inter-reducible with Hilbert’s tenth problem over rationals — a fundamental long standing open problem.

In this chapter we consider for the first time games with a quantitative class of specifications that is closed under boolean conjunction, disjunction and complement with arbitrary (infinite-memory) strategies. We prove that the problem of determining who
is the winner is undecidable even for a fixed formula with only ten dimensions (while the hardness result for finite-memory strategies was not proved for a fixed condition).

Undecidability for (single-dimensional) mean-payoff games was proved for partial information mean-payoff games [43] and for mean-payoff games that are played over infinite-state pushdown automata [38] (see Chapter 7). These works did not exploit the different properties of limit-infimum-average and limit-supremum-average. To the best of our knowledge, the undecidability proof in the chapter is the first to exploit these properties.

**Structure of this chapter.** In the next section we introduce several abbreviations for the formal definitions of multidimensional mean-payoff games. We prove undecidability by a reduction from the halting problem of a two-counter machine. For this purpose we first present a reduction from the halting problem of a one-counter machine and then we extend it to two-counter machine. In Section 6.2 we present the reduction and give some intuition about its correctness. In Section 6.3 we give a formal proof for the correctness of the reduction and extend the reduction to two-counter machine. In Section 6.4 we discuss the key elements of our proof and apply the undecidability result for the similar model of mean-payoff expressions.

### 6.1 Notations and Definitions

For an infinite sequence or reals $x_1, x_2, x_3, \ldots$ we have $\text{LimInfAvg}(x_1, x_2, \ldots) = -\text{LimSupAvg}(-x_1, -x_2, \ldots)$. Hence, an equivalent formulation for multidimensional mean-payoff condition is a positive boolean formula over the atoms $\text{LimInfAvg}_i \sim \nu_i$ and $\text{LimSupAvg}_i \sim \nu_i$ for $\sim \in \{\geq, >\}$. In the sequel we abbreviate $\text{LimInfAvg}_i$ with $\hat{i}$ and $\text{LimSupAvg}_i$ with $\overline{i}$. In this chapter we prove undecidability for the general case and for this purpose it is enough to consider only the $\geq$ operator and thresholds 0. Hence, in the sequel, whenever it is clear that the threshold is 0, we abbreviate the condition $\hat{i} \geq 0$ with $\hat{i}$ and $\overline{i} \geq 0$ with $\overline{i}$. For example, $\hat{i} \lor \overline{j} \land \ell$ stands for $\text{LimInfAvg}_i \geq 0 \lor \text{LimSupAvg}_j \geq 0 \land \text{LimInfAvg}_\ell \geq 0$. By further abuse of notation we abbreviate the current total weight in dimension $i$ by $\hat{i}$ (and make sure that the meaning of $i$ is always clear from the context) and the absolute value of the total weight by $|\hat{i}|$.

### 6.2 Reduction from the Halting Problem and Informal Proof of Correctness

In this chapter we prove the undecidability of determining the winner in games over general multidimensional mean-payoff condition by a reduction from the halting problem of two-counter machine. For this purpose we will first show a reduction from the halting problem of a one-counter machine to multidimensional mean-payoff games, and the reduction from two-counter machines relies on similar techniques. We first give a formal definition for a one-counter machine, and in order to simplify the proofs we give a non-standard definition that is tailored for our needs. A two-sided one-counter machine $M$ consists of two finite set of control states, namely $Q$ (left states) and $P$.
(right states), an initial state \( q_0 \in Q \), a final state \( q_f \in Q \), a finite set of left to right instructions \( \delta_{\ell \rightarrow r} \) and a finite set of right to left instructions \( \delta_{r \rightarrow \ell} \). An instruction determines the next state and manipulates the value of the counter \( c \) (and initially the value of \( c \) is 0). A left to right instruction is of the form of either:

- \( q : \text{if } c = 0 \text{ goto } p \text{ else } c := c - 1 \text{ goto } p' \), for \( q \in Q \) and \( p, p' \in P \); or
- \( q : \text{goto } p \), for \( q \in Q \) and \( p \in P \) (the value of \( c \) does not change).

A right to left instruction is of the form of either

- \( p : c := c + 1 \text{ goto } q \), for \( p \in P \) and \( q \in Q \); or
- \( p : \text{goto } q \), for a state \( p \in P \) and a state \( q \in Q \) (the value of \( c \) does not change).

We observe that in our model, decrement operations are allowed only in left to right instructions and increment operations are allowed only in right to left instructions. However, since the model allows state transitions that do not change the value of the counter (nop transitions), it is trivial to simulate a standard one-counter machine by a two-sided counter machine.

For the reduction we use the states of the game graph to simulate the states of the counter machine and we use two dimensions to simulate the value of the one counter. In the most high level view our reduction consists of three main gadgets, namely, reset, sim and blame (see Figure 6.1), and a state \( q_f \) that represents the final state of the counter machine. Intuitively, in the sim gadget player 1 simulates the counter machine, and if the final state \( q_f \) is reached then player 1 loses. If player 2 detects that player 1 does not simulate the machine correctly, then the play goes to the blame gadget. From the blame gadget the play will eventually arrive to the reset gadget. This gadget assigns proper values for all the dimensions of the game that are suited for an honest simulation in the sim gadget. When a play leaves the reset gadget, it goes to the first state of the simulation gadget which represent the first state of the counter machine.

We now describe the construction with more details. We first present the winning objective and then we describe each of the three gadgets. For a two-sided counter machine \( M \) we construct a game graph with 8 dimensions denoted by \( \ell, r, g_x, c_+, c_-, g_c, x \) and \( y \) and the objective

\[
(\ell \land r \lor g_x) \land (c_+ \land c_- \lor g_c) \land x \land y
\]
6.2.0.1 The sim gadget

In the sim gadget player 1 suppose to simulate the run of $M$, and if the simulation is not honest, then player 2 activates a blame gadget. The simulation of the states is straightforward (since the game graph has states), and the difficulty is to simulate the counter value, more specifically, to simulate the zero testing of the counter. For this purpose we use the dimensions $r, \ell, g_s$ and $c_+, c_-, g_c$.

We first describe the role of $r, \ell$ and $g_s$. The reset gadget makes sure that in every invocation of the sim gadget, we have

$$\text{Avg}(g_s) \approx -1, \text{Avg}(r) \approx 1 \text{ and } \text{Avg}(\ell) \approx 0$$

(The reader should read $a \approx b$ as the value of $a$ is very close to the value of $b$. Precise definitions are given in Section 6.3) Then, during the simulation the value of $g_s$ is always negative, and the blame gadget makes sure that player 1 must play in such a way that whenever the machine $M$ is in a right state,

$$r \approx |g_s| \text{ and } \ell \approx 0$$

and whenever the machine is in a left state, then

$$r \approx 0 \text{ and } \ell \approx |g_s|$$

Intuitively, the role of $\ell$ and $r$ is to make sure that every left to right or right to left transition is simulated by a significant number of rounds in the sim gadget, and $g_s$ is a guard dimension that makes sure that the above assumptions on $r$ and $\ell$ are satisfied.

We now describe the role of $c_+, c_-$ and $g_c$. In the beginning of each simulation (i.e., every time that the sim gadget is invoked), we have

$$\text{Avg}(c_+) \approx \text{Avg}(c_-) \approx 1 \text{ and } \text{Avg}(g_c) \approx -1$$

During the entire simulation we have $\text{Avg}(g_c) \approx -1$ and if $c$ is the value of the counter in the current simulation (i.e., since the sim gadget was invoked), then

$$c_+ \approx |g_c| + |g_s|c \text{ and } c_- \approx |g_c| - |g_s|c$$

Intuitively, whenever $c > 0$, then $c_- \ll |g_c|$, and if $c < 0$ (this can happen only if player 1 is dishonest), then $c_+ \ll |g_c|$ (the reader should read $a \ll b$ as $a$ is much smaller than $b$).

We now describe the gadgets that simulate the operations of inc, dec and nop. The gadgets are illustrated in Figures 6.2-6.5 and the following conventions are used: (i) Player 1 owns the $\bigcirc$ vertices, player 2 owns the $\Box$ vertices, and the $\boxbox$ vertex stands for a gadget; (ii) A transition is labeled either with $a \leftarrow b$ symbol or with a text (e.g., blame). For a transition $e$ the label $a \leftarrow b$ stands for $w_a(e) = b$. Whenever the weight
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE

of a dimension is not explicitly presented, then the weight is 0. We use text labels only to give intuition on the role of the transition. In such transitions the weights of all dimensions are 0.

In order to satisfy the invariants, in the first state of every inc, dec or nop gadget, in a left to right transition, player 1 always moves to the state below (namely, to $\ell \ll 0$?) until $\ell \approx 0$ and $r \approx |g_s|$, and in a right to left transition he always moves to the state below (namely, to $r \ll 0$?) loops until $r \approx 0$ and $\ell \approx |g_s|$. If in a left to right gadget the loop is followed too many times, then $\ell$ is decremented too many times and player 2 has an incentive invokes the $\ell \ll 0$ gadget. If the loop was not followed enough times, then $r$ was not incremented enough times and player 2 invokes the $r \ll |g_s|$ blame gadget. Hence, the blame gadgets allows player 2 to blame player 1 for violating the assumptions about the values of $\ell, r$ and $g_s$.

A transition $q : \text{if } c = 0 \text{ goto } p \text{ else } c := c - 1 \text{ goto } p'$, for $q \in Q$ and $p, p' \in P$ is described in Figure 6.6.

6.2.0.2 The blame gadgets

The role of the blame gadgets is to make sure that the assumptions on $\ell, r$ and $g_s$ are kept in the simulation and to make sure that the zero testing is honestly simulated. There are six blame gadgets. Four for the honest simulation of $r, \ell$ and $g_s$, and two for the zero testing (one for $c > 0$ and one for $c < 0$). The gadgets are described in Figures 6.7-6.12. In the blame $r \ll 0$ and blame $\ell \ll 0$ gadgets the play immediately
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE

\[
\begin{align*}
 r &\leftarrow 1, \ell \leftarrow -1 \\
c_+ &\leftarrow 0, c_- \leftarrow 2 \\
g_c &\leftarrow -1,
\end{align*}
\]

Figure 6.4: dec \( \ell \to r \) gadget.

\[
\begin{align*}
 r &\leftarrow -1, \ell \leftarrow 1 \\
c_+ &\leftarrow 2, c_- \leftarrow 0 \\
g_c &\leftarrow -1,
\end{align*}
\]

Figure 6.5: inc \( r \to \ell \) gadget.

\[
\begin{align*}
\text{blame } r &\ll 0 \\
\text{blame } \ell &\ll 0
\end{align*}
\]

\[
\begin{align*}
\text{blame } \ell &\ll 0 \\
\text{blame } r &\ll 0
\end{align*}
\]

Figure 6.6: \( q \): if \( c = 0 \) then goto \( p \) else \( c := c - 1 \) goto \( p' \)
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE

### 6.2.0.3 The reset gadget

The role of the reset gadget is to assign the following values for the dimensions:

\[
\text{Avg}(\ell) \approx 0, \text{Avg}(r) \approx 1, \text{Avg}(g_s) \approx -1, \text{Avg}(c_-) \approx \text{Avg}(c_+) \approx 1, \text{Avg}(g_c) \approx -1
\]

The gadget is described in Figure 6.13. We construct the gadget is such way that each of the players can enforce the above values (player 2 by looping enough times on the first state, player 1 by looping enough time on his two states). But the construction only gives this option to the players and it does not punish a player if he acts differently. However, the game graph is constructed in such way that if:

- $M$ does not halt and in the reset gadget, at least one of the players, correctly
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE 81

Figure 6.13: Reset gadget.

resets the values, then player 1 wins.

- \( M \) halts and in the reset gadget (at least one of the players) correctly reset the values, then player 2 wins.

Hence, if \( M \) does halts, then player 2 winning strategy will make sure that the reset assigns correct values, and if \( M \) does not halt, then we can rely on player 1 to reset the values. We note that player 2 will not stay forever in his state (otherwise he will lose). In order to make sure that player 1 will not stay forever in one of his states we introduce two liveness dimensions, namely \( x \) and \( y \). In the simulation and blame gadgets they get 0 values. But if player 1 remains forever in one of his two states in the reset gadget, then either \( x \) or \( y \) will have negative \( \limsup \) value and player 1 will lose. Hence, in the reset gadget, player 1 should not only reset the values, but also assign a positive value for \( y \) and then a positive value for \( x \).

6.2.0.4 Correctness of the reduction

We claim that player 1 has a winning strategy if and only if the machine \( M \) does not halt. We first summarize the (informal) invariants that we described in the construction of the reduction. Then, we prove that if \( M \) halts, then player 2 has a winning strategy, and then we prove the converse direction (the proofs are informal, and formal proofs are given in Section 6.3).

**Summary of invariants.** We first describe the reset invariants that hold each time the play leaves the reset gadget (or equivalently, each time the sim gadget is invoked). The reset invariants for the side dimensions are: \( \text{Avg}(g_s) \approx -1, \text{Avg}(r) \approx 1, \text{Avg}(\ell) \approx 0 \), and for the counter dimensions the invariants are: \( \text{Avg}(c_+) \approx 1, \text{Avg}(c_-) \approx 1, \text{Avg}(g_c) \approx -1 \). We now describe the sim invariants that hold whenever the play is in the sim gadget (in all rounds that are played in the sim gadget and also before the first round that is played in the sim gadget). The sim invariants for the side dimensions are: When the play is in a right state (i.e., in a state of \( Q \)) then \( r \approx |g_s| \) and \( \ell \approx 0 \). When the play is in a left state (i.e., in a state of \( P \)) then \( \ell \approx |g_s| \) and \( r \approx 0 \). The next claim follows from the sim invariants: whenever the play is in a state
from $Q$ or $P$ (i.e., after the machine step was simulated), then $c_+ \approx |g_e| + |g_s|c$ and $c_- \approx |g_e| - |g_s|c$, where $c$ is the current value of the counter according to the simulation steps (i.e., $c$ is the value of the number of times the increment gadget was invoked minus the number of times that the decrement gadget was invoked from the beginning of the current invocation of the sim gadget). Informally, the proof of the claim follows by the fact that according to the sim invariants every step of the machine is simulated by a sub-play of length $|g_s|$ and by the fact that in the increment gadget the dimension $c_+$ is incremented by 2 while $|g_s|$ is incremented by 1 (and similar arguments can be applied for the decrement gadget and for dimension $c_-$). We formally prove the claim in Section 6.3.

Another simple consequence of the sim invariants is that $r + \ell \approx |g_s|$ in every round in the sim gadget. Indeed, whenever in a right or left state the equality holds directly from the invariants, and in every transition of the sim gadget the sum of weights of dimension $\ell$ and $r$ is zero.

If $M$ halts, then player 2 wins. The winning strategy for player 2 is as follows: In the reset gadget make sure that the reset invariants are satisfied. This is done by looping the first state of the reset gadget for enough rounds. In the sim gadget, whenever the sim invariants are not fulfilled or whenever player 1 cheats a zero-test, then player 2 invokes a blame gadget. If the sim invariants are fulfilled and player 1 does not cheat a zero test, then it must be the case that the game reaches state $q_f$, and in that case player 2 wins. Otherwise, we claim the player 2 wins.

We first prove that if player 1 violates the sim invariants infinitely often, then the winning condition is violated. W.l.o.g we assume that the first sim invariant is violated infinitely often and the proof for the second invariant is similar. By the assumption infinitely often the play is in a right state and either $r \gg |g_s|$ and $\ell \ll 0$ or $r \ll |g_s|$ and $\ell \gg 0$. If $r \gg |g_s|$, then $\ell \ll 0$ and it follows that in the last round that the state $\ell \ll 0$ was visited, the value of $\ell$ was much smaller than 0. Hence, player 2 invoked the $\ell \ll 0$ blame gadget and the play immediately continued to the reset gadget. If this happens infinitely often then $\ell < 0$ while $g_s < 0$ (as $g_s$ remains negative in the blame $\ell \ll 0$ gadget and never increases in the sim and reset gadgets) and the winning condition is violated. If $r \ll |g_s|$, then player 2 invokes the blame gadget and loop the first state until $r \ll 0$. As $r \ll |g_s|$ we still have $g_s \ll 0$, and thus $r < 0$ while $g_s < 0$ and the condition is violated.

We now assume that the sim invariants are violated only finitely often (for simplicity we assume that they are never violated) and we assume that infinitely often player 1 cheats the zero-test before the play reaches $q_f$. W.l.o.g we assume that player 1 infinitely often declares $c = 0$ while the actual value of $c$ is positive (and the proof for the second cheat is similar). In this case, as $c_- \approx |g_e| - |g_s|c$, we have $c_- \ll |g_s|$. Hence, in the blame $c > 0$ gadget player 2 loops the first state until $c_- \ll 0$. As $c_- \ll |g_s|$ it still holds that $g_e \ll 0$. Hence, $c_- < 0$ while $g_e < 0$ and the condition is violated.

To conclude, if the invariants are not maintained or player 1 does not honestly
simulate the zero-tests, then in each simulation, the guard dimensions have negative average weights, while at least one of the dimensions \( \ell, r, c_+ \) or \( c_- \) has a negative average weight in the blame gadget. Hence, we get that \( g_s, g_c < 0 \) and \( \ell < 0 \) or \( r < 0 \) or \( c_- < 0 \) or \( c_+ < 0 \). Hence, the winning condition is not satisfied and player 2 is the winner.

**If \( M \) does not halt, then player 1 wins.** The winning strategy is to honestly simulate \( M \) while maintaining the *sim invariants* and the *reset invariants*. If player 2 never invokes the blame gadget, namely, the play stays forever in the sim gadget, then the winning condition is satisfied. Indeed, in the sim gadget \( g_s, c_+, c_-, x \) and \( y \) are never decremented, thus their mean-payoff value is at least zero and the winning condition is satisfied. Otherwise, after every invocation of the blame gadget, if a *side blame gadget* was invoked, then either the average value of \( r \) and \( \ell \) is non-negative or the value of the guard dimension \( g_s \) is non-negative. Indeed, if the sim invariants are maintained, then before a blame \( \ell \ll |g_s| \) gadget is invoked we have \( \ell \approx |g_s| \). Hence, if in the gadget we have \( \ell < 0 \), then it must be the case that \( g_s \geq 0 \). Thus, eventually, we get that \( r, \ell \geq 0 \) or \( g_s \geq 0 \). Similarly, when a \( c > 0 \) gadget is invoked, we have \( c = 0 \) and thus \( c_- \approx |g_c| \), and thus in the gadget either \( c_- \) is non-negative or \( g_c \) is non-negative (and similar arguments hold for the \( c < 0 \) gadget and for \( c_+ \)). Hence, we get that \( c_+, c_- \geq 0 \) or \( g_c \geq 0 \). Thus, the winning condition is satisfied, and player 1 is the winner.

### 6.3 Detailed Proof

In the previous section we accurately described the reduction, and only the proof of the correctness was informal. In this section we give a precise proof for the correctness of the reduction, namely, we formally describe player-2 winning strategy in the case that \( M \) halts (Subsection 6.3.1), and player-1 winning strategy in the case that \( M \) does not halt (Subsection 6.3.2). In Subsection 6.3.3 we extend the reduction to two-counter machine.

**Terminology.** In the next subsections we use the next terminology and definitions:

- A *round* is a round in the game graph (i.e., either player-1 or player-2 move).

- A *simulation step* denotes all the rounds that are played in a transition gadget (i.e., in a *nop, inc* or *dec* gadget). Formally, a simulation step is a sub-play that begins and ends in a node from \( P \cup Q \) (i.e., a left or a right state) and visits exactly one time in a left state and exactly one time in a right state.

- A *simulation session* is a sub-play that begins in an invocation of the sim gadget and ends before (or exactly when) the play leaves the sim gadget. The *first \( i \) simulation steps* of a simulation session is a sub-play that begins in an invocation of the sim gadget and ends after \( i \) simulation steps were played.

- A *loop* in a transition gadget is a two round sub-play in the gadget that consists of the loop that is formed by the first state and the state beneath it.

The *total number of rounds* is the total number of rounds (moves) from the beginning of the play. We say that the average weight of dimension \( d \) in round \( i \) is \( a \), and we denote \( \text{Avg}(d) = a \), if the value of dimension \( d \) in round \( i \) is \( a \cdot i \) (i.e., the average
weight of \( d \) from the beginning of the play up to round \( i \) is \( a \). Given a play prefix of length \( i \), we say player-2 can achieve \( \text{Avg}(d_1) \leq a_1 \) while maintaining \( \text{Avg}(d_2) \leq a_2 \), for dimensions \( d_1, d_2 \) and thresholds \( a_1, a_2 \), if player 2 has a strategy to extend the play prefix in such way that in some round \( j \geq i \) it holds that \( \text{Avg}(d_1) \leq a_1 \) and in every round \( k \) such that \( i \leq k \leq j \) it holds that \( \text{Avg}(d_2) \leq a_2 \).

6.3.1 If \( M \) halts, then player 2 is the winner

In this subsection we assume that \( M \) halts. We denote by \( N \) the number of steps after which \( M \) halts (for initial counter value 0) and we denote \( \delta = \frac{1}{(N+1)^2} \). WLOG we assume that \( N > 10 \). The strategy of player 2 in the reset gadget is to achieve the following reset invariants (after the play leaves the gadget):

- \( \text{Avg}(g_s), \text{Avg}(g_c) \leq -\frac{1}{2} \)
- \( (1 - \frac{\delta}{2})|g_s| \leq r \leq (1 + \frac{\delta}{2})|g_s| \)
- \( -\frac{\delta}{2} |g_s| \leq \ell \leq \frac{\delta}{2} |g_s| \)
- \( (1 - \frac{\delta}{2})|g_c| \leq c_+ \leq c_+ \leq (1 + \frac{\delta}{2})|g_c| \)

We note that player 2 can maintain the above by looping sufficiently long time in the first state, and once the invariants are reached, player 1 cannot violate them in his states in the reset gadget (since the average value of \( g_s \) and \( g_c \) can only get closer to \(-1\), the value of \( \frac{r}{|g_s|} \) only gets closer to 0 and \( \frac{\ell - \delta}{|g_s|} \) and \( \frac{c_+}{|g_s|} \) only gets closer to 1).

The strategy of player 2 in the sim gadget is to maintain, in every step of the simulation session, the next three invariants, which we denote by the left right invariants:

- (Left state invariant) If the machine is in a left state, then \( (1 - \epsilon)|g_s| \leq \ell \leq (1 + \epsilon)|g_s| \) and \( -\epsilon|g_s| \leq r \leq \epsilon|g_s| \).
- (Right state invariant) If the machine is in a right state, then \( (1 - \epsilon)|g_s| \leq r \leq (1 + \epsilon)|g_s| \) and \( -\epsilon|g_s| \leq \ell \leq +\epsilon|g_s| \).
- (Minimal value invariant) In every round of a simulation session \( r, \ell \geq -\epsilon|g_s| \).

We denote \( \delta = \frac{1}{N+1+2N(1+2\epsilon)} \). We first prove that under these invariants \( \text{Avg}(g_s) \leq -\delta \) in every round of the play. Then we use this fact to show that if player 1 violates these invariants, then player 2 can violate \( (\ell \wedge x \vee g_s) \), and therefore he wins.

**Lemma 37.** Assume that for a given simulation session: (i) in the beginning of the session \( \text{Avg}(g_s) \leq -\frac{1}{2} \); (ii) no more that \( N \) steps are played in the simulation session; and (iii) the left-right invariants are maintained in the session. Then for every round in the session \( \text{Avg}(g_s) \leq -\delta \).

**Proof.** We denote by \( R \) the number of rounds that were played before the current invocation of the simulation gadget. We claim that after simulating \( i \) steps of the machine (in the current invocation of the sim gadget), the total number of rounds in
the play (i.e., number of rounds from the beginning of the play, not from the beginning of the current invocation) is at most \( R + 2i \cdot |g_s|(1 + 2\epsilon) \). The proof is by a simple induction, and for the base case 1 = 0 the proof is trivial. For \( i > 0 \), we assume WLOG that the \( i \)-th transition is a left-to-right transition. Hence, before the last simulation step we had \( r \geq -\epsilon|g_s| \) and after the \( i \)-th step was completed we had \( r \leq (1 + \epsilon)|g_s| \). Since in every odd round of a step gadget the value of \( r \) is incremented by 1, we get that at most \( 2(1 + 2\epsilon)|g_s| \) rounds were played and the proof of the claim follows (and the proof for a right-to-left transition is symmetric).

Hence, after \( N \) simulation steps we have

\[
\text{Avg}(g_s) \leq \frac{g_s}{R + 2N|g_s|(1 + 2\epsilon)}
\]

Since in the beginning of the sim gadget we had \( \text{Avg}(g_s) \leq -\frac{1}{2} \), then \( R \leq \frac{|g_s|}{2} \). Hence, and since \( g_s < 0 \) we get

\[
\text{Avg}(g_s) \leq \frac{|g_s| + 2N|g_s|(1 + 2\epsilon)}{2N(1 + 2\epsilon)} = -\frac{1}{2} + 2N(1 + 2\epsilon) = -\delta
\]

We note that in every transition of a simulation session the value of \( g_s \) is not changed. Hence, \( \text{Avg}(g_s) \) gets the maximal value after the \( N \)-th step and the proof is complete.

**Lemma 38.** Let \( \gamma = \min\left(\frac{\epsilon^2}{4}, \frac{1}{1 + \frac{\delta}{2\epsilon} - \frac{\epsilon}{2}}\right) \). If player 1 violates the left-right invariants in the first \( N \) steps of a simulation session, then player 2 can achieve in the blame gadget either \( \text{Avg}(r) \leq -\gamma \) or \( \text{Avg}(\ell) \leq -\gamma \) (or both) while maintaining \( \text{Avg}(g_s), \text{Avg}(g_c) \leq -\gamma \).

**Proof.** We first prove the assertion over the value of \( g_c \). It is an easy observation that if at the invocation of the sim gadget \( \text{Avg}(g_c) \leq -\frac{1}{2} \), then it remains at most \( -\frac{1}{2} \) as it gets a value of \( -1 \) in every round in a blame gadget and \( -1 \) in every odd round in a step gadget.

Next, we prove the assertion for the left-state and minimal value invariants and the proof for the right-state invariant is symmetric. Recall that the invariant consistencies of four assumptions, namely, (i) \((1 - \epsilon)|g_s| \leq \ell \) after a right to left transition; (ii) \( \ell \leq (1+\epsilon)|g_s| \) after a right to left transition; (iii) \(-\epsilon|g_s| \leq r \) in every round; and (iv) \( r \leq \ell \) after a right to left transition. We first prove the assertion when the first condition is violated, i.e., we assume that \( \ell < (1 - \epsilon)|g_s| \). If this is the case after a right-to-left transition, then player 2 will invoke the blame gadget after the transition ends. In the blame gadget he will traverse the self-loop for \( X \cdot (1 - \frac{\epsilon}{2}) \) times, where \( X \) is the value of \( |g_s| \) before the invocation of the blame gadget, and then he will go to the reset gadget. As a result (since in every loop \( \ell \) is decremented by 1 and \( g_s \) is incremented by 1) we get that the value of \( \ell \) and \( g_s \) is at most \( -X \cdot \frac{\epsilon}{2} \). Before the last simulation step the left-right invariants were maintained. Hence, before the last step we had \( \ell \geq -\epsilon|g_s| \) (by the left-right invariants) and thus the last step had at most \( |g_s| \) rounds (as we assume
that after the last step $\ell < (1 - \epsilon)|g_s|$. In addition, as the invariants were maintained, by Lemma 37 we get that before the last step we had $\text{Avg}(g_s) \leq -\delta$ and thus after the last step we have $\text{Avg}(g_s) \leq -\frac{\delta}{2}$ (as the value of $g_s$ is not changed in simulation steps).

Hence, if $R$ is the number of rounds before the invocation of the blame gadget, then $R \leq \frac{X}{25}$. Hence, after the blame gadget ends, we have

$$\text{Avg}(\ell), \text{Avg}(g_s) \leq -\frac{X \cdot \frac{\epsilon}{1 + \frac{\epsilon}{2}}}{R + X \cdot (1 - \frac{\epsilon}{2})} \leq -\frac{X \cdot \frac{\epsilon}{1 + \frac{\epsilon}{2}}}{\frac{X}{25} + X \cdot (1 - \frac{\epsilon}{2})} = -\frac{\frac{\epsilon}{2}}{1 + \frac{1}{25} - \frac{\epsilon}{2}}$$

In addition, the value of $g_s$ is incremented in every round of the blame gadget. Thus, if after the gadget ends we have $\text{Avg}(g_s) \leq -\gamma$, then in every round in the blame gadget we also have $\text{Avg}(g_s) \leq -\gamma$.

If the second condition is violated, namely, if $\ell > (1 + \epsilon)|g_s|$, then we claim that it must be the case that $r < -\frac{|g_s|}{2}$. Indeed, when the sim gadget is invoked we have $r \leq |g_s|(1 + \frac{\epsilon}{2})$ and $\ell \leq |g_s|\frac{\epsilon}{1}$. In the sim gadget the value of the sum $r + \ell$ is not changed (since $r$ is incremented only when $\ell$ is decremented and vice versa). Hence, the sum never exceeds $|g_s|(1 + \frac{\epsilon}{2})$. Thus, if $\ell > (1 + \epsilon)|g_s|$, then it must be the case that $r < -\frac{|g_s|}{2}$. Hence, in the first round that $\text{Avg}(r) \leq -\frac{\epsilon}{2}|g_s|$ player 2 can invoke the blame gadget which leads the play to the reset gadget after exactly one move. We note that in this scenario the left-right invariants are satisfied and thus, after leaving the blame gadget by Lemma 37 we have $\text{Avg}(g_s) \leq -\delta$ and as $r < -\frac{|g_s|}{2}$ we get that $\text{Avg}(r) \leq -\frac{\epsilon}{2}$.

If the third condition is violated, namely, if $r < -\epsilon|g_s|$, then it must be the case that the condition is first violated in a left to right transition (since in a right to left transition $r$ is incremented) and the proof follows by the same arguments as in the proof of the second case.

Finally, if the fourth condition is violated, namely, if $r > \epsilon|g_s|$, then by analyzing the sum $r + \ell$ we get that $\ell \leq (1 - \frac{\epsilon}{2})|g_s|$. We repeat the same analysis as in the case where the first invariant is violated (i.e., when $\ell \leq (1 - \epsilon)|g_s|$) and get that

$$\text{Avg}(g_s), \text{Avg}(r) \leq -\frac{\frac{\epsilon}{2}}{1 + \frac{1}{25} - \frac{\epsilon}{4}}.$$

The proof is complete. \hfill \Box

By Lemma 38, if player 2 maintains the reset invariant in the reset gadget, then other in finitely many simulation sessions, in every simulation session player 1 must satisfy the left-right invariants. Otherwise, we get that infinitely often the average value of either $r$ or $\ell$ is at most $-\gamma$ while the average value of $g_s$ is always at most $-\gamma$ (indeed, by Lemma 38 $\text{Avg}(g_s) \leq -\gamma$ in the blame gadgets, by Lemma 37 $\text{Avg}(g_s) \leq -\delta \leq -\gamma$ in the sim gadget, and in the reset gadget $g_s$ always gets a value of $-1$). Hence $g_s < 0$ and either $r < 0$ or $\ell < 0$ and thus the condition $(\ell \land r \lor \ell \land r)$ is violated and therefore player 1 is losing.
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE

In the next three lemmas we prove that player 1 must honestly simulates the zero-testing. The first lemma is a simple corollary of the left-right invariants.

**Lemma 39.** Under the left-right invariants, in the dec, inc and nop gadgets, player 1 follows the loop of the first state at most $|g_s|(1 + 2\epsilon)$ times and at least $|g_s|(1 - 2\epsilon)$ times.

**Proof.** Upper bound: in a left to right transition, we have $\ell \leq |g_s|(1 + \epsilon)$ before the transition and $\ell \geq -|g_s|\epsilon$ after the transition, and in every loop $\ell$ is decremented by 1.

Lower bound: in a left to right transition, we have $\ell \geq |g_s|(1 - \epsilon)$ before the transition and $\ell \leq |g_s|\epsilon$ after the transition, and in every loop $\ell$ is decremented by 1.

The proof for a right to left transition is symmetric. □

The next lemma shows the correlation between $g_c$ and $c_+$ and $c_-$.

**Lemma 40.** Let $\#\text{inc}$ (resp., $\#\text{dec}$) be the number of times that the inc (dec) gadget was visited (in the current simulation session), and we denote $c = \#\text{inc} - \#\text{dec}$ (namely, $c$ is the actual value of the counter in the counter machine $M$). Then under the left-right invariants, in the first $N$ steps of the simulation session we always have $c_+ \leq |g_c|(1 + \epsilon) + c|g_s| + \frac{|g_s|}{2}$ and $c_- \leq |g_c|(1 - \epsilon) - c|g_s| + \frac{|g_s|}{2}$.

**Proof.** We prove the claim of the lemma for $c_+$ and the proof for $c_-$ is symmetric. Let $X$ be the value of $|g_c|$ when the sim gadget is invoked. By the reset invariants we get that $c_+ \leq X(1 + \frac{\epsilon}{4})$. By Lemma 39 we get that every visit in the inc gadget contributes at most $|g_s|(1 + 2\epsilon)$ more to $c_+$ than its contribution to $|g_c|$ and every visit in the dec gadget contributes at least $|g_s|(1 - 2\epsilon)$ more to $|g_c|$ than its contribution to $c_+$. Hence,

$$c_+ \leq X(1 + \frac{\epsilon}{4}) + (|g_c| - X) + \#\text{inc} \cdot |g_s|(1 + 2\epsilon) - \#\text{dec} \cdot |g_s|(1 - 2\epsilon) = |g_c| + \epsilon X + (\#\text{inc} - \#\text{dec})|g_s|(1 + 2\epsilon) + 4\epsilon|g_s| \cdot \#\text{dec}$$

We recall that $c = (\#\text{inc} - \#\text{dec})$, and observe that $X \leq |g_c|$, and that $\#\text{dec} \leq N$ and thus $\epsilon \cdot \#\text{dec} < \frac{1}{10}$. Hence, we get that $c_+ \leq |g_c|(1 + \epsilon) + c|g_s| + \frac{|g_s|}{2}$. □

The next lemma suggests that player 1 must honestly simulate the zero-tests.

**Lemma 41.** If the reset and left-right invariants hold, then for $\gamma = \min\left(\frac{1}{20N}, \frac{\epsilon}{2}\right)$ the following hold: (i) if the blame $c < 0$ gadget is invoked and $c < 0$ then player 2 can achieve $\text{Avg}(c_+) \leq -\gamma$ while maintaining $\text{Avg}(g_s), \text{Avg}(g_c) \leq -\gamma$; and (ii) if the blame $c > 0$ gadget is invoked and $c > 0$ then player 2 can achieve $\text{Avg}(c_-) \leq -\gamma$ while maintaining $\text{Avg}(g_s), \text{Avg}(g_c) \leq -\gamma$.

**Proof.** We prove the first item of the lemma and the proof for the second item is symmetric. Suppose that $c < 0$ (i.e., $c \leq -1$) when blame $c < 0$ gadget is invoked. Let $X$ and $Y$ be the values of $|g_c|$ and $|g_s|$ before the invocation of the blame gadget. Then by Lemma 40, before the invocation we have $c_+ \leq X(1 + \epsilon) - \frac{Y}{2}$. Hence, by traversing the loop of the first state of the blame $c < 0$ gadget for $X(1 + \epsilon) - \frac{Y}{4}$
times we get $c_+ \leq -\frac{y}{4}$ and $g_c \leq \epsilon X - \frac{y}{4}$. Let $R$ be the number of rounds that were played from the beginning of the play (and not just from the beginning of the current invocation of the sim gadget). Since $g_c$ is decremented by at most 1 in every round we get that $X(1 + \epsilon) - \frac{y}{4} \leq 2X \leq 2R$. By lemma 37 we have $\frac{y}{4} \leq -\delta$. Hence, $\text{Avg}(c_+) \leq \frac{c_+}{2R} \leq -\frac{y}{8R} \leq -\frac{\delta}{8}$. Similarly, since $\frac{x}{R}$ is bounded by 1, we have $\text{Avg}(g_c) \leq \epsilon X - \frac{\delta}{8} \leq \frac{\epsilon}{2} - \frac{\delta}{8}$. Recall that $\delta = \frac{1}{2} + 2N(1+2\epsilon)$. Hence, $\text{Avg}(g_c) \leq \frac{2\epsilon+4N\epsilon+8\epsilon^2-1}{8(\frac{1}{2}+N(1+2\epsilon))}$ and since $\epsilon = \frac{1}{N(1+1)^2}$ and $N > 10$ we get that $\text{Avg}(g_c) \leq -\frac{1}{20N}$. Note that $g_c$ is only incremented in the blame gadget. The value of $g_s$ was at most $-\delta R$ before the blame gadget, and in the blame gadget $g_s$ is decreased by 1 in every round. Hence $\text{Avg}(g_s) \leq -\delta$ in every round of the blame gadget and the proof follows by taking $\gamma = \min(\frac{1}{20N}, \frac{\delta}{8})$. \\We are now ready to prove one side of the reduction.\\

**Proposition 1.** If the counter machine $M$ halts, then player 2 has a winning strategy for violating $(\ell^c \land \bar{\ell}^c \land \bar{\gamma}^c) \land (c_+ \land c_- \land \bar{g}_c) \land \bar{x} \land \bar{y}$. Moreover, if $M$ halts then for $\zeta = \min(\gamma, 1)$ player 2 has a winning strategy for violating $(\ell \geq -\zeta \land r \geq -\zeta \land \bar{\ell} \geq -\zeta) \land (c_+ \geq -\zeta \land c_- \geq -\zeta \land \bar{g}_c \geq -\zeta) \land \bar{x} \geq -\zeta \land \bar{y} \geq -\zeta$ (where $\gamma$ is the positive constant from Lemma 38).

**Proof.** Suppose that $M$ halts and let $N$ be the number of steps that $M$ runs before it halts (for an initial counter value 0). Player-2 strategy is to

- Maintain the reset-invariants.
- Whenever the left-right invariants are violated, he invokes a side blame gadget.
- Whenever the zero-testing is dishonest, he activates the corresponding blame gadget (either $r > 0$ or $c < 0$).
- If $q_f$ is reached, he stays there forever.

The correctness of the construction is immediate by the lemmas above. We first observe that it is possible for player 2 to satisfy the reset-invariants and that if player 1 stays in the reset gadget forever, then he loses.

Whenever the left-right invariant is violated, then the average weight of $r$ and/or $\ell$ is negative, while the average weight of $g_c$ and $g_c$ remains negative. Hence, if in every simulation session player 1 violates the left-right invariants in the first $N$ steps we get that the condition is violated since $\bar{g}_c \leq -\gamma$ and either $x \leq -\gamma$ or $\ell \leq -\gamma$. Hence, we may assume that these invariants are kept in every simulation session.

Whenever the zero-testing is dishonest (while the left-right invariants are satisfied), then by Lemma 41, player 2 can invoke a counter blame gadget and achieve negative average for either $c_+$ or $c_-$ while maintaining $g_c$ and $g_s$ negative. If in every simulation session player 1 is dishonest in zero-testing, then we get that either $c_- \leq -\gamma$ or $c_+ \leq -\gamma$. (**UNDECIDABLE**)}
while $\overline{yc} \leq -\gamma$ and the condition is violated. Hence, we may assume that player 1 honestly simulates the zero-tests. Finally, if the transitions of $M$ are properly simulated, then it must be the case the state $q_f$ is reached and when looping this state forever player 1 loses (since $\overline{\pi} \leq -1 < 0$).

### 6.3.2 If $M$ does not halt, then player 1 is the winner

Suppose that $M$ does not halt. A winning strategy of player 1 in the reset gadget is as following: Let $i$ be the number of times that the reset gadget was visited, and we denote $\epsilon_i = \frac{1}{1+4i}$. Similarly to player-2 strategy in Subsection 6.3.1, player-1 strategy in the reset gadget is to achieve the following invariants (after the play leaves the gadget):

- $\text{Avg}(g_s), \text{Avg}(g_c) \leq -\frac{1}{2}$
- $(1 - \frac{\epsilon_i}{4})|g_s| \leq r \leq (1 + \frac{\epsilon_i}{4})|g_s|$
- $\frac{\epsilon_i}{4} \leq \ell \leq \frac{\epsilon_i}{4} |g_s|$
- $(1 - \frac{\epsilon_i}{4})|g_c| \leq c_+, c_- \leq (1 + \frac{\epsilon_i}{4})|g_c|$

To satisfy these invariants, he follows the self-loop of his first state until $\text{Avg}(y) \geq 0$ and then follows the self-loop of the second state until the invariants are fulfilled and $\text{Avg}(x) \geq 0$. In the sim gadget, player-1 strategy is to simulate every $\text{nop}, \text{inc}$ and $\text{dec}$ step by following the self-loop in the corresponding gadget for $|g_s|$ rounds. In addition, player 1 honestly simulates the zero-tests.

We denote the above player-1 strategy by $\tau$ and next lemma shows the basic properties of a play according to $\tau$.

**Lemma 42.** In any play according to $\tau$, after the reset gadget was visited for $i$ times, in the sim gadget we always have: (i) in a right state: $r \geq -\epsilon_i |g_s|, \ell \geq (1 - \epsilon_i) |g_s|$ and in a left state $\ell \geq -\epsilon_i |g_s|, r \geq (1 - \epsilon_i) |g_s|$; (ii) in every round of the simulation session $r, \ell \geq -\epsilon_i |g_s|$; and (iii) $c_+ \geq (1 - \epsilon_i) |g_c| + c |g_s|$ and $c_- \geq (1 - \epsilon_i) |g_c| - c |g_s|$, where $c = \#\text{inc} - \#\text{dec}$ in the current invocation of the sim gadget.

**Proof.** The proof of the first item is straight forward. Initially, when the sim gadget is invoked (and starts in a left state) we have $-\epsilon_i |g_s| \leq \ell \leq \epsilon_i |g_s|$ and $(1 - \epsilon_i) |g_s| \leq r \leq (1 + \epsilon_i) |g_s|$. In every left to right transition $r$ is decremented by $|g_s|$ and $\ell$ is incremented by $|g_s|$. In every right to left transition $\ell$ is decremented by $|g_s|$ and $r$ is incremented by $|g_s|$. Hence, the proof of the first and second item follows.

In order to prove the third item we denote by $K$ the value of $|g_c|$ when the sim gadget is invoked. Initially we have $c_+ \geq (1 - \epsilon_i) K$. After every simulation step we have $c_+ \geq (1 - \epsilon_i) K + |g_s| (2 \#\text{inc} + \#\text{nop})$ and $|g_c| = K + |g_s| (\#\text{inc} + \#\text{dec} + \#\text{nop})$. Hence, $c_+ \geq (1 - \epsilon_i) |g_c| + c |g_s|$. The proof for $c_-$ is symmetric.

We will use the next lemma to prove that player 2 is losing when he invokes the blame gadgets.
Lemma 43. In a play prefix consistent with \( \tau \):

1. In the blame \( \ell \ll 0 \) and blame \( r \ll 0 \) gadgets: in the (one) round that is played in the gadget \( \text{Avg}(\ell), \text{Avg}(r) \geq -\epsilon_i \).

2. In the blame \( \ell \ll |g_s| \) gadget: in every round, if \( \text{Avg}(\ell) \leq -\epsilon_i \), then \( \text{Avg}(g_s) \geq -\epsilon_i \).

3. In the blame \( r \ll |g_s| \) gadget: in every round, if \( \text{Avg}(r) \leq -\epsilon_i \), then \( \text{Avg}(g_s) \geq -\epsilon_i \).

4. In the blame \( c < 0 \) gadget: in every round, if \( \text{Avg}(c_+) \leq -\epsilon_i \), then \( \text{Avg}(g_c) \geq -\epsilon_i \).

5. In the blame \( c > 0 \) gadget: in every round, if \( \text{Avg}(c_-) \leq -\epsilon_i \), then \( \text{Avg}(g_c) \geq -\epsilon_i \).

Where \( i \) is the number of times that the reset gadget was visited.

Proof. Proof of item 1: By Lemma 42 it is always hold that in the sim gadget \( r, \ell \geq -\epsilon_i|g_s| \). Hence, when the \( \ell \ll 0 \) or blame \( r \ll 0 \) gadgets are invokes the assertion still holds. As the average weight of \( g_s \) is at most \(-1\) we get that \( \text{Avg}(r), \text{Avg}(\ell) \geq -\epsilon_i \).

Proof of item 2: By Lemma 42 we know that when blame \( \ell \ll |g_s| \) gadget is invoked, the value of \( \ell \) is at least \((1 - \epsilon_i)|g_s| \). Let \( R \) be the total number of rounds that were played before the blame gadget was invoked, and by \( Y \) the value of \(|g_s| \) before the invocation of the blame gadget. In order to obtain \( \text{Avg}(\ell) \leq -\epsilon_i \), player 2 should follow the self-loop of the gadget at least \( X = \frac{(1-\epsilon_i)Y + R\epsilon_i}{1-\epsilon_i} \) rounds. In every such round the value of \( g_s \) is incremented by 1. We note that \( X \geq Y \). Hence, after \( X \) rounds we have \( g_s = -Y + X \geq 0 \) and in particular \( \text{Avg}(g_s) \geq -\epsilon_i \).

Proof of item 3: Symmetric to the proof of item 2.

Proof of item 4: By Lemma 43 we know that before the invocation of the blame gadget we have \( c_+ \geq (1 - \epsilon_i)|g_c| + c|g_s| \). Since \( \tau \) honestly simulates the zero-tests we get that \( c \geq 0 \). Hence, \( c_+ \geq (1 - \epsilon_i)|g_c| \). Thus, by the same arguments as in the proof of item 2 we get that if \( \text{Avg}(c_+) \leq -\epsilon_i \), then \( \text{Avg}(g_c) \geq 0 \geq -\epsilon_i \).

Proof of item 4: By Lemma 43 we know that before the invocation of the blame gadget we have \( c_- \geq (1 - \epsilon_i)|g_c| - c|g_s| \). Since \( \tau \) honestly simulates the zero-tests we get that \( c = 0 \). The rest of the proof is symmetric to the proof of item 3. \( \square \)

We are now ready to prove the \( \tau \) is a winning strategy.

Proposition 2. If \( M \) does not halt, then \( \tau \) is a winning strategy.

Proof. In order to prove that \( \tau \) satisfies the condition \((\ell \wedge x \vee \overline{g_s}) \land (c_+ \wedge c_- \vee \overline{g_c}) \land \overline{x} \land \overline{y} \) it is enough to prove that when playing according to \( \tau \), for any constant \( \delta > 0 \) the condition \((\ell \geq -\delta \land x \geq -\delta \lor \overline{g_s} \geq -\delta) \land (c_+ \geq -\delta \land c_- \geq -\delta \lor \overline{g_c} \geq -\delta) \land \overline{x} \land \overline{y} \) is satisfied.

Let \( \delta > 0 \) be an arbitrary constant and in order to prove the claim we consider two distinct cases.
CHAPTER 6. MULTIDIMENSIONAL MEAN-PAYOFF GAMES ARE UNDECIDABLE

In the first case, player 2 strategy will invoke the blame gadgets only finitely many times. Hence, either there is a suffix that is played only in a blame gadget or only in a reset gadget (and in such suffixes player 2 loses) or there is an infinite suffix that is played only in the sim gadget (as \( M \) does not halt). In the sim gadget the values of \( x, y \) and \( g_s \) are not changed. Hence the long-run average weight of these dimensions is 0. In addition, \( c_+ \) and \( c_- \) are never decremented in the sim gadget. Hence, their long-run average weight is also at least 0. Thus, the condition is satisfied.

In the second case we consider, player 2 always eventually invokes a blame gadget. Since a blame gadget is invoked infinitely many times we get that the reset gadget is invoked infinitely often, and thus \( \overline{x}, \overline{y} \geq 0 \). In addition, the sim gadget is invoked infinitely often. Let \( i \) be the minimal index for which \( \epsilon_i \leq \delta \). We claim that after the \( i \)-th invocation of the sim gadget, in every round (i) either \( \text{Avg}(\ell) \geq -\epsilon_i \wedge \text{Avg}(r) \geq -\epsilon_i \) or \( \text{Avg}(g_s) \geq -\epsilon_i \); and (ii) either \( \text{Avg}(c_+) \geq -\epsilon_i \wedge \text{Avg}(c_-) \geq -\epsilon_i \) or \( \text{Avg}(g_c) \geq -\epsilon_i \). In order to prove the claim we show that the assertions hold in the beginning of the sim gadget, and that if the assertions hold in the beginning of each gadget, then they hold in all rounds that are played until the play leaves the gadget. We first analyze the reset gadget, then the sim gadget and finally the blame gadgets. In the reset gadget \( \ell, r, c_+ \) and \( c_- \) get non-negative weights and \( g_s \) and \( g_c \) get \(-1\) weight in every round. Hence, if the assertions hold in the beginning of the gadget, then they hold in the entire sub-play that is played in the gadget. In the sim gadget, by the reset invariants, the assertions hold in the invocation of the gadget. In the rest of the rounds the assertion for \( \ell \) and \( r \) hold by Lemma 42, which shows that \( r, \ell \geq -\epsilon_i |g_s| \), and from the fact that \(|\text{Avg}(g_s)|\) is always bounded by 1 (as the weights of \( g_s \) are bounded by \( \pm 1 \)). The assertion for \( c_+ \) and \( c_- \) follows from the fact that in a simulation session, these dimensions have only non-negative weights. Finally, in the blame gadgets, the assertions hold due to Lemma 43, and the proof of the claim is complete. Thus, as of certain round, either \( \text{Avg}(\ell) \) and \( \text{Avg}(r) \) are always at least \(-\epsilon_i \), or infinitely often \( \text{Avg}(g_s) \geq -\epsilon_i \). Hence, \((\ell \geq -\epsilon_i \wedge r \geq -\epsilon_i \vee g_s \geq -\epsilon_i)\) is satisfied and similarly \((c_+ \geq -\epsilon_i \wedge c_- \geq -\epsilon_i \vee g_c \geq -\epsilon_i)\) is satisfied. Therefore, since \( \epsilon_i \leq \delta \), we get that \( \tau \) satisfies \((\ell \geq -\delta \wedge r \geq -\delta \vee g_s \geq -\delta) \wedge (c_+ \geq -\delta \wedge c_- \geq -\delta \vee g_c \geq -\delta) \wedge \overline{\tau} \wedge \overline{g_c} \). \( \square \)

6.3.3 Extending the reduction to two-counter machine

When \( M \) is a two-counter machine, we use 4 dimensions for the counters, namely \( c_1^+, c_1^-, c_2^+, c_2^- \) and one guard dimension \( g_c \). The winning condition is \((\ell \wedge r \vee g_s) \wedge (c_1^+ \wedge c_1^- \wedge c_2^+ \wedge c_2^- \vee g_c) \wedge \overline{\tau} \wedge \overline{g_c} \). In a \textit{nop} gadget all four dimensions \( c_1^+, c_1^-, c_2^+, c_2^- \) get a value of 1 in the self-loop. When a counter \( c_i \) (for \( i = 1, 2 \)) is incremented (resp., decremented), then counter \( c_i^+ \) and \( c_i^- \) are assigned with weights according to the weights of \( c_+ \) and \( c_- \) in the \textit{inc} (\textit{dec}) gadget that we described in the reduction for a one counter machine, and \( c_1^{i+1}, c_2^{i+1} \) are assigned with weights according to a \textit{nop} gadget.

The proofs of Proposition 1 and Proposition 2 easily scale to a two-counter machine. Hence, the undecidability result is obtained.
THEOREM 9. The problem of deciding who is the winner in a multidimensional mean-payoff game with ten dimensions is undecidable.

6.4 Discussion

Additional intuition. Previous undecidability results for mean-payoff games relied on a reduction from the universality problem of a non-deterministic weighted automaton over finite-words. In such setting, the input for the automaton is a run of a counter machine and the automaton verifies that the run is valid. The non-determinism allows the automaton to decide after the run ends whether (i) it simulates the first counter or the second counter; and (ii) whether a dishonest zero-test was done for a positive counter or for a zero counter.

In our reduction player 2 must identify dishonest simulation during the run and he does not know in advance whether a dishonest zero-test will be done for the first counter or for the second counter. Hence, if we would apply the standard technique, then one run that is dishonest for the first counter might fix the mean-payoff of a previous run that was dishonest for the second counter. For this purpose we introduce the reset gadget that in a way clears the effect of previous simulation sessions. However, with the reset gadget the effect of the rounds that are played in the simulation gadget becomes neglectable. For this purpose we introduce the notion of left-to-right and right-to-left transitions which make sure that every simulation step takes a non-neglectable fraction of the run. Both the reset gadget and the blame gadgets crucially rely on the combination of limit-supremum-average and limit-infimum-average objectives. Indeed, games over the condition \( (\ell \land r \lor g_s) \land (c_1^+ \land c_1^- \land c_2^+ \lor g_c) \land x \land y \) or the condition \( (\ell \land r \lor g_s) \land (c_1^+ \land c_1^- \land c_2^+ \lor g_c) \land x \land y \) are decidable [99].

Undecidability for Similar Objectives Alur et al. [7] considered the same objectives as in this chapter. Boker et al. [21] and Tomita et al. [93] extended Alur et al objectives with boolean temporal logic. These works only considered one-player games and did not settle the decidability of two-player games. Our undecidability result trivially implies that two-player games over their objectives are undecidable.

Chatterjee et al. [31] considered mean-payoff expressions that assign a (one-dimensional) real value to the vectors \( \text{LimInfAvg}(\pi) \) and \( \text{LimSupAvg}(\pi) \). As the objective is quantitative and not boolean, there are three problem that are relevant to games for a given expression \( E \): (i) Can player 1 assure \( E \geq \nu \)?; (ii) Can player 1 assure \( E > \nu \)?; and (iii) What is the maximal (supremum) value the player 1 can assure? In the previous chapter we proved that when player 1 is restricted to finite-memory strategies, then the decidability of the first problem is inter-reducible with Hilbert’s Tenth problem over rationals (a long standing open problem), but the second problem is decidable and the third problem is computable. In the next theorem we show that the winning condition that we use in the undecidability proof can be encoded as a mean-payoff expression and conclude that games over mean-payoff expressions with arbitrary strategies are undecidable.
Theorem 10. The problem of deciding who is the winner of two-player games over mean-payoff expressions are undecidable.

Proof. Consider the construction from Section 6.2 and the expressions

\[ E_1 = \max (\min(\text{LimInfAvg}_\ell, \text{LimInfAvg}_r), \text{LimSupAvg}_g), \]

\[ E_2 = \max (\min(\text{LimInfAvg}_{c_1}, \text{LimInfAvg}_{c_1^-}, \text{LimInfAvg}_{c_2}, \text{LimInfAvg}_{c_2^-}), \text{LimSupAvg}_{g_c}), \]

\[ E_3 = \min(\text{LimSupAvg}_x, \text{LimSupAvg}_y), \]

\[ E = \min(E_1, E_2, E_3) \] and \[ F = -E, \]

and the boolean formula \( \varphi = (\ell \land r \lor \overline{g}) \land (c_1 \land c_1^- \land c_2 \land c_2^- \lor \overline{g_c}) \land \overline{\pi} \land \overline{\overline{\pi}}. \) It is a trivial observation that for any play \( \pi \): (i) \( \varphi \) is satisfied iff \( E(\pi) \geq 0 \); and (ii) \( \varphi \) is not satisfied iff \( F(\pi) > 0 \). Hence, we have a reduction from determining whether player 1 wins for multidimensional mean-payoff games to mean-payoff expression games with weak inequality, and a reduction from determining whether player 2 wins for multidimensional mean-payoff games to mean-payoff expression games with strong inequality. Moreover, by Proposition 1 it follows that if player 1 cannot satisfy \( E \geq 0 \), then for some \( \zeta > 0 \) he cannot satisfy \( E \geq -\zeta \). Hence if we could compute the maximal (supremum) value that player 1 can assure we could determine whether he wins for \( E \geq 0 \) (if the value is negative, then he loses, and otherwise he wins). \( \square \)
Part II

Mean-Payoff Pushdown Games
The study of two-player finite-state games with qualitative objectives has been extended in two orthogonal directions in the literature: (1) two-player infinite-state games with qualitative objectives; and (2) two-player finite-state games with quantitative objectives. One of the most well-studied models of infinite-state games with qualitative objectives is pushdown games (or games on recursive state machines) that can model reactive systems with recursion (or model the control flow of sequential programs with recursion). Pushdown games with reachability and parity objectives have been studied in [8, 9, 101, 102] (also see [23, 24, 49, 50] for sample research in stochastic pushdown games). The most well-studied quantitative objective is the mean-payoff objective, where a reward is associated with every transition and the goal of one of the players is to maximize the long-run average of the rewards (and the goal of the opponent is to minimize). Two-player finite-state games with mean-payoff objectives have been studied in [47, 70, 110], and more recently applied in synthesis of reactive systems with quality guarantee [16] and robustness [17]. Moreover recently many quantitative logics and automata theoretic formalisms have been proposed with mean-payoff objectives in their heart to express properties such as reliability requirements, and resource bounds of reactive systems [21, 34, 44]. Thus pushdown games with mean-payoff objectives would be a central theoretical question for model checking of quantitative logics (specifying reliability and resource bounds) on reactive systems with recursion feature.

**Pushdown mean-payoff games.** In this part we study for the first time pushdown games with mean-payoff objectives (to the best of our knowledge mean-payoff objectives have not been studied in the context of pushdown games). In pushdown games two types of strategies are relevant and studied in the literature. The first are the *global* strategies, where a global strategy can choose the successor state depending on the entire global history of the play (where history is the finite sequence of configurations of the current prefix of a play). The second are the *modular* strategies, and modular strategies are understood more intuitively in the model of games on recursive state machines. A recursive state machine (RSM) consists of a set of component machines (or modules). Each module has a set of nodes (atomic states) and boxes (each of which is mapped to a module), a well-defined interface consisting of entry and exit nodes, and edges connecting nodes/boxes. An edge entering a box models the invocation of the module associated with the box and an edge leaving the box represents return from the module. In the game version the nodes are partitioned into player-1 nodes and player-2 nodes. Due to recursion the underlying global state-space is infinite and isomorphic to pushdown games. The polynomial-time equivalence of pushdown games and recursive games has been established in [9]. A modular strategy is a strategy that has only local memory, and thus, the strategy does not depend on the context of invocation of the module, but only on the history within the current invocation of the module. In other words, modular strategies are appealing because they are stackless strategies, decomposable into one for each module. In this work we will study pushdown games with mean-payoff objectives for both global and modular strategies.
Previous results. Pushdown games with qualitative objectives were studied in [101, 102]. It was shown in [102] that solving pushdown games (i.e., determining the winner in pushdown games) with reachability objectives under global strategies is EXPTIME-hard, and pushdown games with parity objectives under global strategies can be solved in EXPTIME. Thus it follows that pushdown games with reachability and parity objectives under global strategies are EXPTIME-complete. The notion of modular strategies in games on recursive state machines was introduced in [8, 9]. It was shown that the modular strategies problem is NP-complete in pushdown games with reachability and parity objectives in general [8, 9]. The results of [9] also presents more refined complexity results in terms of the number of exit nodes, showing that if every module has single exit, then the problem is polynomial for reachability objectives [9] and in NP \( \cap \) coNP for parity objectives [8].

Our contributions. In this work we present a complete characterization of the computational and strategy complexity of pushdown games and pushdown systems (one-player pushdown games or pushdown automata) with mean-payoff objectives. Solving a pushdown system (resp. pushdown game) with respect to a mean-payoff objective is to decide whether there exists a path that (resp. a winning strategy to ensure that every path possible given the strategy) satisfies the mean-payoff objective. Our main results for computational complexity are as follows.

1. **Global strategies.** We show that pushdown systems (one-player pushdown games) with mean-payoff objectives under global strategies can be solved in polynomial time, whereas solving pushdown games with mean-payoff objectives under global strategies is undecidable. We also consider multidimensional pushdown mean-payoff with conjunctive condition and show that the complexity of solving one-player game is polynomial.

2. **Modular strategies.** Solving pushdown systems with single exit nodes with mean-payoff objectives under modular strategies is NP-hard, and pushdown games with mean-payoff objectives under modular strategies can be solved in NP. Thus both pushdown systems and pushdown games with mean-payoff objectives under modular strategies are NP-complete. We also consider multidimensional pushdown mean-payoff with conjunctive condition and show that two-player games are undecidable.

First observe that our hardness result for modular strategies is different from the NP-hardness of [9] because the hardness result of [9] shows hardness for games with reachability objectives and requires that the number of modules with multiple exit nodes are not bounded (in fact if every module of the recursive game has a single exit, then the problem is in PTIME for reachability and in NP \( \cap \) coNP for parity objectives). In contrast we show that for mean-payoff objectives the problem is NP-hard even for pushdown systems (only one player), where every module has a single exit node, under modular strategies. Second, we also observe the very different complexity of global
and modular strategies for mean-payoff objectives in pushdown systems vs pushdown games: the global strategies problem is computationally inexpensive (in PTIME) as compared to the modular strategies problem (which is NP-complete) in pushdown systems; whereas the global strategies problem is computationally infeasible (undecidable) as compared to the modular strategies problem (which is NP-complete) in pushdown games. Also observe that in contrast to finite-state game graphs where the complexities for mean-payoff and parity objectives match, for pushdown systems and games, the complexities of parity and mean-payoff objectives are very different. Along with the computational complexities, we also establish the optimal strategy complexity showing that global winning strategies for mean-payoff objectives in general require infinite memory even in pushdown systems; whereas memoryless or positional (independent of history) strategies suffice for modular strategies for mean-payoff objectives in pushdown games. Finally we also study the stack boundedness conditions where the goal of one player along with maximizing the mean-payoff objective is also to ensure that the height of the stack is bounded. We show that all the complexities for the additional stack boundedness condition along with mean-payoff objectives are the same in pushdown systems and games as without the stack boundedness condition.

**Organization.** In Chapter 7 we study mean-payoff pushdown games with global strategies, and in Chapter 8 we study games with modular strategies over the recursive state machines.

**Bibliographic note.** The results of this part were first published in:

- Krishnendu Chatterjee, Yaron Velner: Mean-Payoff Pushdown Games. LICS 2012.

- Krishnendu Chatterjee, Yaron Velner: Hyperplane Separation Technique for Multidimensional Mean-Payoff Games. CONCUR 2013.
Chapter 7

Mean-Payoff Pushdown Games

7.1 Mean-Payoff Pushdown Graphs

In this section we consider pushdown graphs (or pushdown systems) with mean-payoff objectives. We start with an overview of the solution, and the basic notions of stack alphabet and commands.

Overview of the solution. We first characterize the pumpable paths in a pushdown graph that determine the possible mean-payoff values of the graph. In a finite-state graph a path with non-negative mean-payoff exists if and only if there is a finite pumpable path (namely, a cycle) with non-negative weight. For infinite-state graphs, and pushdown graphs in particular, the latter does not hold. However we show that a path with a strictly positive mean-payoff exists if and only if there is a finite pumpable path with positive weight. For this purpose we first characterize the pumpable paths in a pushdown graph and in Section 7.1.1 we obtain a polynomial algorithm to detect pumpable paths with a positive weight, and hence we get a polynomial-time algorithm to detect a path with a positive mean-payoff in a pushdown graph. In Section 7.1.2 we show a reduction from the problem of detecting paths with non-negative mean-payoff to the problem of detecting paths with positive mean-payoff and a polynomial-time algorithm for mean-payoff pushdown graphs is obtained.

Stack alphabet and commands. Let Γ denote a finite set of stack symbols (called the stack alphabet), and \( \text{Com}(\Gamma) = \{ \text{skip}, \text{pop} \} \cup \{ \text{push}(z) \mid z \in \Gamma \} \) denote the set of stack commands over Γ. Intuitively, the command \( \text{skip} \) does nothing, \( \text{pop} \) deletes the top element of the stack, \( \text{push}(z) \) puts \( z \) on the top of the stack. For a stack command \( \text{com} \) and a stack string \( \alpha \in \Gamma^+ \) we denote by \( \text{com}(\alpha) \) the stack string obtained by executing the command \( \text{com} \) on \( \alpha \).

Weighted pushdown systems. A weighted pushdown system (WPS) (or a weighted pushdown graph) is a tuple:

\[
\mathcal{A} = \langle Q, \Gamma, q_0 \in Q, E \subseteq (Q \times \Gamma) \times (Q \times \text{Com}(\Gamma)), w : E \rightarrow \mathbb{Z} \rangle,
\]

where \( Q \) is a finite set of states with \( q_0 \) as the initial state; \( \Gamma \) is a finite stack alphabet and we assume that there is a special initial stack symbol \( \perp \in \Gamma \); \( E \) describes the set of edges or transitions of the pushdown system; and \( w \) is a weight function that assigns
integer weights to every edge (and the weights are encoded in binary). We assume that \( \bot \) can be neither put nor removed from the stack. A configuration of a WPS is a pair \((\alpha, q)\) where \( \alpha \in \Gamma^+ \) is a stack string and \( q \in Q \). For a stack string \( \alpha \) we denote by Top(\( \alpha \)) the top symbol of the stack. The initial configuration of the WPS is \((\bot, q_0)\).

We use \( W \) to denote the maximal absolute weight of the edge weights.

**Successor configurations, paths, and ultimately periodic paths.** Given a WPS \( A \), a configuration \( c_{i+1} = (\alpha_{i+1}, q_{i+1}) \) is a successor configuration of a configuration \( c_i = (\alpha_i, q_i) \), if there is an edge \((q_i, \gamma_i, q_{i+1}, \text{com}) \in E\) such that \( \text{com}(\alpha_i) = \alpha_{i+1} \), where \( \gamma_i = \text{Top}(\alpha_i) \). A path \( \pi \) is a sequence of configurations. A path \( \pi = \langle c_1, \ldots, c_{n+1} \rangle \) is a valid path if for all \( 1 \leq i \leq n \) the configuration \( c_{i+1} \) is a successor configuration of \( c_i \) (and the notation is similar for infinite paths). In the sequel we shall refer only to valid paths. Let \( \pi = \langle c_1, c_2, \ldots, c_i, c_{i+1}, \ldots \rangle \) be a path. We denote by \( \pi[j] = c_j \) the \( j \)-th configuration of the path and by \( \pi[i_1, i_2] = \langle c_{i_1}, c_{i_1+1}, \ldots, c_{i_2} \rangle \) the segment of the path from the \( i_1 \)-th to the \( i_2 \)-th configuration. A path can equivalently be defined as a sequence \( \langle c_1 e_1 c_2 e_2 \ldots e_n \rangle \), where \( c_1 \) is the initial configuration and \( e_i \) are valid transitions.

A path \( \pi \) is ultimately periodic if there exists a finite sequence \( \xi_1 \in E^* \) and a non-empty finite sequence \( \xi_2 \in E^+ \) of transitions such that the path consists of \( \xi_1 \) followed by \( \xi_2 \) forever, i.e., \( \pi = \langle c_1 \xi_1 (\xi_2)^\omega \rangle \). A configuration \( c_r \) is reachable if there is a finite path that begins at the initial configuration and ends in \( c_r \). Similarly, a path \( \pi \) is reachable if its initial configuration is reachable.

**Average weights of paths.** For a finite path \( \pi \), we denote by \( w(\pi) \) the sum of the weights of the edges in \( \pi \) and \( \text{Avg}(\pi) = \frac{w(\pi)}{|\pi|} \), where \(|\pi|\) is the length of \( \pi \), denotes the average of the weights. For an infinite path \( \pi \), we denote by \( \text{LimSupAvg}(\pi) \) (resp. \( \text{LimInfAvg}(\pi) \)) the limit-sup (resp. limit-inf) of the averages (long-run average or mean-payoff objectives), i.e., \( \lim \sup(\text{Avg}(\pi[0, i])) \) and \( \lim \inf(\text{Avg}(\pi[0, i])) \). We say that \( \pi \) is a positive path if \( w(\pi) > 0 \), and negative, non-negative and non-positive paths are similarly defined.

**Notations.** We shall use (i) \( \gamma \) or \( \gamma_i \) for an element of \( \Gamma \); (ii) \( e \) or \( e_i \) for a transition (equivalently an edge) from \( E \); (iii) \( \alpha \) or \( \alpha_i \) for a string from \( \Gamma^+ \). For a path \( \pi = \langle c_1, c_2, \ldots \rangle = \langle c_1 e_1 c_2 \ldots \rangle \) we denote by (i) \( q_i \): the state of configuration \( c_i \), and (ii) \( \alpha_i \): the stack string of configuration \( c_i \).

**Stack height and additional stack height of paths.** For a path \( \pi = \langle (\alpha_1, q_1), \ldots, (\alpha_n, q_n) \rangle \), the stack height of \( \pi \) is the maximal height of the stack in the path, i.e., \( \text{SH}(\pi) = \max\{|\alpha_1|, \ldots, |\alpha_n|\} \). The additional stack height of \( \pi \) is the additional height of the stack in the segment of the path, i.e., the additional stack height \( \text{ASH}(\pi) \) is \( \text{SH}(\pi) - \max\{|\alpha_1|, |\alpha_n|\} \).

**Pumpable pair of paths.** Let \( \pi = \langle c_1 e_1 c_2 \ldots \rangle \) be a finite or infinite path. A pumpable pair of paths for \( \pi \) is a pair of non-empty sequence of edges: \( \langle p_1, p_2 \rangle = \langle e_{i_1} e_{i_1+1} \ldots e_{i_1+n_1}, e_{i_2} e_{i_2+1} \ldots e_{i_2+n_2} \rangle \), for \( n_1, n_2 \geq 0, i_1 \geq 0 \) and \( i_2 > i_1 + n_1 \) such that for every \( j \geq 0 \) the path \( \pi_j^{(p_1, p_2)} \) obtained by pumping the pair \( p_1 \) and \( p_2 \) of paths \( j \)
Proof. We first select a subpath of \( \pi \), denoted by \( \pi^* \), such that \( \pi^* = (c_1^*, \ldots, c_p^*, \ldots, c_n^*) \) and the following conditions hold: (i) \( c_1^* \) is a local minimum in \( \pi^* \), (ii) \( |\alpha_i^*| = |\alpha_i^*| \), and (iii) \( |\alpha_i^*| = |\alpha_i^*| + d \). The subpath is selected as follows: consider a configuration \( c_1^* \) in \( \pi \) where the stack height is maximal, and \( c_1^* \) is the closest configuration before \( c_p^* \) where the stack height is exactly \( d \) less than the stack height of \( c_p^* \), and similarly \( c_n^* \) is the closest configuration after \( c_p^* \) where the stack height is exactly \( d \) less than that of \( c_p^* \) (see Figure 7.1). Clearly all the three conditions are satisfied. Let \( c_j^* \) (resp. \( c_j'' \)) be the closest configuration before (resp. after) \( c_p^* \) such that the stack height of \( c_j^* \) (resp. \( c_j'' \)) is \( |\alpha_i^*| + j \), for \( j \geq 0 \). Since \( d \geq |(Q \cdot |\Gamma|)|^2 \) it follows from the pigeonhole principle that there exist \( j_1, j_2 \) such that \( j_1 > j_2 \), and the states and the top stack symbol of \( c_j^* \), and \( c_j'' \) are identical, and the states and the top stack symbol of \( c_{j_2}'' \) and \( c_{j_2}' \) are identical. It is straightforward to verify that the sequence \( p_1 \) of edges between \( c_{j_1}' \) and \( c_{j_2}' \) along
with the sequence $p_2$ of edges between $e''_{j_2}$ and $e''_{j_1}$ form a pumpable pair, i.e., $(p_1, p_2)$ is a pumpable pair for $\pi$.

In the following lemma we establish the connection of additional stack height and the existence of a pumpable pair of paths with positive weights.

**Lemma 45.** Let $c_1$ and $c_2$ be two configurations. If there exists $n \in \mathbb{Z}$ such that the minimal additional stack height (denoted by $d$) of the paths from $c_1$ to $c_2$ with weight at least $n$ is at least $(|Q| \cdot |\Gamma|)^2$, then there exists a path from $c_1$ to $c_2$ with additional stack height $d$ that contains a pumpable pair $(p_1, p_2)$ with $w(p_1) + w(p_2) > 0$.

**Proof.** Let $\pi$ be one of the shortest paths (in terms of length) from $c_1$ to $c_2$ with weight at least $n$ and additional stack height $d$. By Lemma 44, the path $\pi$ has a pumpable pair $(p_1, p_2)$ of paths. Assume towards a contradiction that the weight of the pair is not positive (i.e., consider that $w(p_1) + w(p_2) \leq 0$). Then, we can remove the pair and obtain a path $\pi' = \pi'[p_1, p_2]$ from $c_1$ to $c_2$ with $w(\pi') \geq w(\pi)$. The path $\pi'$ is shorter in length than $\pi$, since either $p_1$ or $p_2$ is not an empty path. Moreover, the additional stack height of $\pi'$ is at most $d$. This yields a contradiction to the fact that both $d$ and the length of $\pi$ are minimal.

**Mean-payoff objectives with strict and non-strict inequalities.** For a given integer $r$, the mean-payoff objective $\text{LimInfAvg} \bowtie r$ (resp. $\text{LimSupAvg} \bowtie r$) defines the set of infinite paths $\pi$ such that $\text{LimInfAvg}(\pi) \bowtie r$ (resp. $\text{LimSupAvg}(\pi) \bowtie r$), where $\bowtie \in \{\geq, >\}$. Mean-payoff objectives with integer threshold $r$ can be transformed to threshold 0 by subtracting $r$ from all transition weights. Hence in this work w.l.o.g we will consider the mean-payoff objectives (i) $\text{LimInfAvg} > 0$ (resp. $\text{LimSupAvg} > 0$), and call them mean-payoff objectives with strict inequality; and (ii) $\text{LimInfAvg} \geq 0$ (resp. $\text{LimSupAvg} \geq 0$), and call them mean-payoff objectives with non-strict inequality. We are interested in solving WPSs with mean-payoff objectives, i.e., to decide if there is a path that satisfies the objective.

**Example 6.** We now illustrate with an example that the witness paths for non-strict inequality mean-payoff objectives are not necessarily ultimately periodic. Consider the
WPS $A$ with two states $Q = \{q_1, q_2\}$, with two symbol stack alphabet $\Gamma = \{\bot, \gamma\}$, and the edge set $E = \{e_1, e_2, \ldots, e_5\}$ is described as follows: $e_1 = (q_1, \bot, q_1, \text{push}(\gamma)), e_2 = (q_1, \gamma, q_1, \text{push}(\gamma)), e_3 = (q_1, \gamma, q_2, \text{skip}), e_4 = (q_2, \gamma, q_2, \text{pop}),$ and $e_5 = (q_2, \bot, q_1, \text{skip})$. The weight function is as follows: $w(e_4) = 1$, and all other edge weights are $-1$. (See Figure 7.2 for a pictorial description). For $i \geq 1$, consider the path segment $\rho_i = c_1 e_1 e_2^{i-1} e_3 e_4 e_5$ that executes the edge $e_1$, followed by $(i-1)$-times the edge $e_2$, then the edge $e_3$, followed by $i$-times the edge $e_4$, and finally the edge $e_5$. It is straightforward to verify that for the infinite path $\pi = (\bot, q_1) \rho_1 \rho_2 \rho_3 \ldots$ we have that $\text{LimSupAvg}(\pi) = \text{LimInfAvg}(\pi) = 0$. However for every valid path $\pi = c_1 \xi_1 (\xi_2)^\omega$, where $\xi_1 \in E^*$ and $\xi_2 \in E^+$ it must be the case that either (i) $\xi_2 = e_2$ and then $\text{LimInfAvg}(\pi) = \text{LimSupAvg}(\pi) = -1$ or that (ii) $\xi_2$ is a cycle with length at most $|\xi_2|$ and has weight at most $-1$, and hence $\text{LimInfAvg}(\pi) \leq \text{LimSupAvg}(\pi) \leq -\frac{1}{|\xi_2|} < 0$.

Ignoring finite plays. For technical convenience, we will only consider infinite plays, and consider that finite plays do not satisfy the mean-payoff objective. Thus if there are no transitions from a state, then we consider it as a loosing sink state (a state with a self-loop with negative weight).

7.1.1 Objectives $\text{LimInfAvg} > 0$ and $\text{LimSupAvg} > 0$

In this section we consider limit-average (or mean-payoff) objectives with strict inequality. We show that WPSs with such objectives can be solved in polynomial time. A crucial concept in the proof is the notion of good cycles, and we define them below.

**Good cycle.** A finite path $\pi = \langle c_1, \ldots, c_n \rangle$ is a good cycle if the following conditions hold:

1. $w(\pi) > 0$ (the weight of the path is positive); and
2. $\pi$ is a non-decreasing path; and
3. let $c_1 = (\alpha_1, q_1)$ and $c_n = (\alpha_n, q_n)$, then $q_1 = q_n$ and $\text{Top}(\alpha_1) = \text{Top}(\alpha_n)$.

We first prove two lemmas and the intuitive descriptions of them are as follows: In the first lemma (Lemma 46) we show that for every WPS, for every natural number $d$, there exists a natural number $n$ such that if there is a path with weight at least $n$ and additional stack height at most $d$, then there is a good cycle in the WPS. The second
Restrictions to reachable configurations. In the sequel of this whole section we will only consider reachable configurations and reachable paths, and not explicitly mention the reachable property. For good cycles, we often mention the reachable property explicitly.

**Lemma 46.** Let \( A \) be a WPS. For every \( d \in \mathbb{N} \) there exists \( n_{A,d} \in \mathbb{N} \) such that the following assertion holds: If there exists a non-decreasing path \( \pi = \langle c_1, \ldots, c_r \rangle \) such that (i) \( w(\pi) \geq n_{A,d} \) and (ii) \( \text{ASH}(\pi) \leq d \); then \( A \) has a reachable good cycle.

**Proof.** Let \( G_d \) be a graph that contains all the paths that begin in \( c_1 \) and end in \( c_r \) with additional stack height at most \( d \) and for which \( c_1 \) is a local minimum. Note that \( \text{ASH}(\pi) \leq d \) implies that \( \text{SH}(c_r) - \text{SH}(c_1) \leq d \). Hence the graph \( G_d \) is a finite graph (bounded given \( d \)), and the maximal absolute weight is at most \( W \) (the maximal absolute weight of \( A \)). A reachable positive cycle in \( G_d \) implies the existence of a reachable good cycle in \( A \), and if no positive cycle is reachable, then the weight of each path is bounded by \( |G_d + 1| \cdot W \). Thus with \( n_{A,d} = |G_d + 1| \cdot W \) we obtain the desired result.

**Lemma 47.** Let \( A \) be a WPS. Let \( n \in \mathbb{Z} \) and let \( \pi = \langle c_1, \ldots, c_r \rangle \) be a non-decreasing path with weight at least \( n \), with minimal additional stack height among all paths from \( c_1 \) to \( c_r \) with weight at least \( n \). If \( \text{ASH}(\pi) \geq (|Q| \cdot |\Gamma|)^2 \), then for every \( m \in \mathbb{N} \) there exists a non-decreasing path \( \pi_m \) from \( c_1 \) to \( c_r \) with \( w(\pi_m) \geq m \).

**Proof.** By Lemma 45 there exists a path \( \overline{\pi} \) from \( c_1 \) to \( c_r \) that has a pumpable pair \((p_1, p_2)\) such that \( w(p_1) + w(p_2) > 0 \). Hence for every \( i \in \mathbb{N} \) we get that \( w(\overline{\pi}^{i+1}_{(p_1, p_2)}) > w(\overline{\pi}^i_{(p_1, p_2)}) \) (i.e., the weight after pumping \( i + 1 \) times the pair of paths exceeds the weight of pumping \( i \) times). Hence for \( i = m - w(\overline{\pi}) \) we get that \( w(\overline{\pi}^i_{(p_1, p_2)}) \geq m \). The desired result follows.

**Lemma 48.** Let \( A \) be a WPS. There exists \( n_A \in \mathbb{N} \) such that if there exists a non-decreasing path \( \pi \) from configuration \( c_1 \) to configuration \( c_r \) and \( w(\pi) \geq n_A \), then one of the following conditions holds:

1. The WPS \( A \) has a reachable good cycle.

2. For every \( n' \in \mathbb{N} \) there exists a non-decreasing path \( \pi' \) from \( c_1 \) to \( c_r \) with \( w(\pi') > n' \).
Proof. Observe that the number \( n_A \) is of our choice and we will choose it sufficiently large for the proof. Let \( d^* = (|Q| \cdot |\Gamma|)^2 \), and our choice of \( n_A \) is \(|Q| \cdot |\Gamma| \cdot n_{A,d^*} \) (where \( n_{A,d^*} \) is as defined in Lemma 46). Let \( \pi = \langle c_1, c_2, \ldots, c_r \rangle \) be a path such that \( c_1 \) is a local minimum and \( w(\pi) \geq n_A \). Let \( m_1, \ldots, m_j \) be the local minima along the path. Note that \( m_1 = c_1 \) and \( c_r = m_j \). Also note that \( j \geq |\alpha_r| - |\alpha_1| \). Note that if \( m_{i_1} = (\alpha_{i_1}, \gamma, q) \) and \( m_{i_2} = (\alpha_{i_2}, \gamma, q) \) (for some \( \gamma \in \Gamma \)), then if a good cycle does not exist we get that the weight of the path between \( m_{i_1} \) and \( m_{i_2} \) is not positive. Hence, since \( Q \) and \( \Gamma \) are finite, either a good cycle exists (by the pigeonhole principle) or there exists \( m_i, m_{i+1} \) such that \( \alpha_{i+1} = \alpha_i \gamma \) for some \( \gamma \in \Gamma \cup \{\epsilon\} \) (where \( \epsilon \) denotes the empty string) and there exists a path from \( m_i \) to \( m_{i+1} \) such that \( m_i \) is a local minimum and the weight of the path is at least \( n_{A,d^*} \) (since the longest sequence of local minimum configurations that do not contain a cycle is of length at most \(|Q| \cdot |\Gamma|\), and there is a sequence of acyclic configurations that has a weight of at least \( n_A \)). Let \( \pi^* \) be such a path with minimal additional stack height between \( m_i \) and \( m_{i+1} \). We consider two cases to complete the proof.

1. If the additional stack height of \( \pi^* \) is smaller than \( d^* \), then by Lemma 46 we have a reachable good cycle from \( m_i \) and since \( m_i \) is reachable from \( c_1 \) we have reachable good cycle from \( c_1 \) (condition 1 of the lemma holds).

2. If the additional stack of \( \pi^* \) is at least \( d^* \), then by Lemma 47 for every \( n_0 \) we can construct a path \( \pi_{n_0} \) between \( m_i \) and \( m_{i+1} \) with weight \( w(\pi_{n_0}) \) at least \( n_0 \), and \( m_1 \) is a local minimum of \( \pi_{n_0} \). For \( n' \in \mathbb{N} \), let \( n_0 = n' + W \cdot |\pi| \), and let \( \pi' \) be the path constructed using the segment from \( c_1 \) to \( m_i \), then the path \( \pi_{n_0} \), and then the segment of \( \pi \) from \( m_{i+1} \) to \( c_r \). The configuration \( c_1 \) is a local minimum of \( \pi' \) and the weight of \( \pi' \) is at least \( n_0 - W \cdot |\pi| \geq n' \). Hence it follows that for every \( n' \) we can construct a path from \( c_1 \) to \( c_r \) with \( c_1 \) as a local minimum and weight at least \( n' \) (condition 2 of the lemma holds).

This completes the proof of the lemma.

\[ \square \]

Lemma 49. Let \( A \) be WPS. The following statements are equivalent: (i) There exists a path \( \pi_1 \) with \( \text{LimSupAvg}(\pi_1) > 0 \); (ii) there exists a path \( \pi_2 \) with \( \text{LimInfAvg}(\pi_2) > 0 \); and (iii) there exists a path \( \pi \) that contains a good cycle.

Proof. The direction from right to left (i.e., (iii) \( \Rightarrow \) (ii) \( \Rightarrow \) (i)) is immediate. Let \( \pi = \pi_1 \pi_2 \) be a finite path in \( A \) such that \( \pi_1 \) is a good cycle. Let \( \pi_1 = c_1 e_1^1 \ldots e_{n_1}^1 \) and \( \pi_2 = c_2 e_2^2 \ldots e_{n_2}^2 \). The infinite path \( \pi' = \pi_1 c_2 (e_2^2 \ldots e_{n_2}^2)^\omega \) obtained by repeating the good cycle forever is a valid path which witnesses that \( \text{LimSupAvg}(\pi') \geq \text{LimInfAvg}(\pi') > 0 \).

In order to prove the opposite direction, we consider an infinite path \( \pi \) such that \( \text{LimSupAvg}(\pi) > 0 \). Let \( q \in Q \) and \( \gamma \in \Gamma \) be such that the sequence \( m_1 = (\alpha_{i_1}, q) \), \( m_2 = (\alpha_{i_2}, q) \ldots \) is an infinite sequence of local minima of \( \pi \) and \( \text{Top}(\alpha_{i_k}) = \gamma \) (note that such state and symbol are guaranteed to exist due to the existence of infinitely many local
minima and finiteness of $Q$ and $\Gamma$). If there exists $j > 1$ such that $w(\pi[i_1, i_j]) > 0$ then by definition $\pi[i_1, i_j]$ is a good cycle and the result follows. Otherwise let us assume that for every $j > 1$ we have $w(\pi[i_1, i_j]) \leq 0$. As $\limsup_{\text{Avg}}(\pi) > 0$ it follows that for every $n^* \in \mathbb{N}$ there exists $n \in \mathbb{N}$ with $i_n > 1$ such that the path $\pi[i_1, i_n]$ contains a prefix with weight at least $n^*$ (otherwise $\limsup_{\text{Avg}}(\pi) \leq 0$). We now use Lemma 48 to complete the proof. Let $n^* = n_A$ (where $n_A$ is as used in Lemma 48). Let $\pi' = m_1, \ldots, c^*$ be the prefix of $\pi[i_1, i_n]$ such that $w(\pi') \geq n^*$. If the first condition of Lemma 48 holds (i.e., $\mathcal{A}$ has a good cycle), then we are done with the proof. Otherwise, by condition 2 of Lemma 48 it follows that for every $n_0 \in \mathbb{N}$ there exists a path $\pi_{n_0}$ from $m_1$ to $c^*$ such that $n_1$ is a local minimum and $w(\pi_{n_0}) \geq n_0$. Let us choose $n_0 = W \cdot |\pi[i_1, i_n]| + 1$. Then consider the path $\pi = \pi_{n_0} \pi[i + |\pi'|, i_n]$ that is obtained by concatenating the witness path $\pi_{n_0}$ for $n_0$ from $m_1$ to $c^*$, and then the part of $\pi$ from $c^*$ to $\pi[i_n]$. For the path $\pi$ we have (i) the sum of weights is at least $n_0 - W \cdot |\pi[i_1, i_n]| \geq 1 > 0$; (ii) $\pi[i_1]$ is a local minimum; and (iii) the state and the top stack symbol of $\pi[i_1]$ and $\pi[i_n]$ are the same. Thus $\pi$ is a witness good cycle. For conclusion we get that if $\limsup_{\text{Avg}}(\pi) > 0$, then there exists a good cycle, which also implies that there exists a path $\pi'$ such that $\liminf_{\text{Avg}}(\pi') > 0$. This concludes the proof of the lemma.

In the above key lemma we have established the equivalence of the decision problems for WPSs with mean-payoff objectives with strict inequality and the problem of determining the existence good cycles. We will now present a polynomial-time algorithm for detecting good cycles. To this end we introduce the notion of non-decreasing $\alpha$-paths and summary functions.

Non-decreasing $\alpha$-paths. A path from a configuration $(\alpha \gamma, q_1)$ to a configuration $(\alpha \gamma \alpha_2, q_2)$ is a non-decreasing $\alpha$-path if $(\alpha \gamma, q_1)$ is a local minimum. Note that if $\pi$ is a non-decreasing $\alpha$-path for some $\alpha \in \Gamma^*$, then the same sequence of transitions leads to a non-decreasing $\beta$-path for every $\beta \in \Gamma^*$. Hence we say that $\pi$ is a non-decreasing path if there exists $\alpha \in \Gamma^*$ such that $\pi$ is a non-decreasing $\alpha$-path.

Summary function. Let $\mathcal{A}$ be a WPS. For $\alpha \in \Gamma^*$ we define $s_\alpha : Q \times \Gamma \times Q \rightarrow \{-\infty\} \cup \mathbb{Z} \cup \{\omega\}$ as follows.

1. $s_\alpha(q_1, \gamma, q_2) = \omega$ iff for every $n \in \mathbb{N}$ there exists a non-decreasing path from $(\alpha \gamma, q_1)$ to $(\alpha \gamma, q_2)$ with weight at least $n$.

2. $s_\alpha(q_1, \gamma, q_2) = z \in \mathbb{Z}$ iff the weight of the maximal-weight non-decreasing path from configuration $(\alpha \gamma, q_1)$ to configuration $(\alpha \gamma, q_2)$ is $z$.

3. $s_\alpha(q_1, \gamma, q_2) = -\infty$ iff there is no non-decreasing path from $(\alpha \gamma, q_1)$ to $(\alpha \gamma, q_2)$.

Remark 5. For every $\alpha_1, \alpha_2 \in \Gamma^*$: $s_{\alpha_1} \equiv s_{\alpha_2}$.

Due to Remark 5 it is enough to consider only $s \equiv s_\perp$. The computation of the summary function will be achieved by considering the stack height bounded summary functions defined below.
Stack height bounded summary function. For every \( d \in \mathbb{N} \), the stack height bounded summary function \( s_d : Q \times \Gamma \times Q \to \{-\infty\} \cup \mathbb{Z} \cup \{\omega\} \) is defined as follows: (i) \( s_d(q_1, \gamma, q_2) = \omega \) iff for every \( n \in \mathbb{N} \) there exists a non-decreasing path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) with weight at least \( n \) and additional stack height at most \( d \); (ii) \( s_d(q_1, \gamma, q_2) = z \) iff the weight of the maximal-weight non-decreasing path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) with additional stack height at most \( d \) is \( z \); and (iii) \( s_d(q_1, \gamma, q_2) = -\omega \) iff there is no non-decreasing path with additional stack height at most \( d \) from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\).

Basic facts of summary functions. We have the following basic facts: (i) for every \( d \in \mathbb{N} \), we have \( s_{d+1} \geq s_d \) (monotonicity); and (ii) \( s_{d+1} \) is computable in polynomial time from \( s_d \) and \( A \) (we will show this fact in Lemma 52). We first present a lemma that shows that from \( s_d \), with \( d = (|Q| \cdot |\Gamma|)^2 \), we obtain the values of function \( s \) for all values in \( \mathbb{Z} \cup \{-\infty\} \).

**Lemma 50.** Let \( d = (|Q| \cdot |\Gamma|)^2 \). For all \( q_1, q_2 \in Q \) and \( \gamma \in \Gamma \), if \( s(q_1, \gamma, q_2) \in \mathbb{Z} \cup \{-\infty\} \), then \( s(q_1, \gamma, q_2) = s_d(q_1, \gamma, q_2) \).

**Proof.** By definition we have \( s(q_1, \gamma, q_2) \geq s_d(q_1, \gamma, q_2) \). Assume towards a contradiction that \( s(q_1, \gamma, q_2) > s_d(q_1, \gamma, q_2) \), then there exists a non-decreasing path \( \pi \) with minimal additional stack height from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) with weight \( n > s_d(q_1, \gamma, q_2) \) and additional stack height \( d' > (|Q| \cdot |\Gamma|)^2 \). Hence by Lemma 47 for every \( m \in \mathbb{N} \) there exists a non-decreasing path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) with weight at least \( m \) (note that in Lemma 47 the witness path constructed by pumping the positive pumpleable pair yields a non-decreasing path). Hence \( s(q_1, \gamma, q_2) = \omega \) in contradiction to the assumption that \( s(q_1, \gamma, q_2) \in \mathbb{Z} \cup \{-\infty\} \). The desired result follows.

Our goal now is the computation of the \( \omega \) values of the summary function. To achieve the computation of \( \omega \) values we will define another summary function \( s^* \) and a new WPS \( \mathcal{A}^* \) such that certain cycles in \( \mathcal{A}^* \) will characterize the \( \omega \) values of the summary function. We now define the summary function \( s^* \) and the pushdown system \( \mathcal{A}^* \). Let \( d = (|Q| \cdot |\Gamma|)^2 \). The new summary function \( s^* \) is defined as follows: if the values of \( s_d \) and \( s_{d+1} \) are the same then it is assigned the value of \( s_d \), and otherwise the value \( \omega \). Formally, for all states \( q_1, q_2 \in \mathcal{A} \) and a stack symbol \( \gamma \),

\[
s^*(q_1, \gamma, q_2) = \begin{cases} 
  s_d(q_1, \gamma, q_2) & \text{if } s_d(q_1, \gamma, q_2) = s_{d+1}(q_1, \gamma, q_2) \\
  \omega & \text{if } s_d(q_1, \gamma, q_2) < s_{d+1}(q_1, \gamma, q_2).
\end{cases}
\]

The new WPS \( \mathcal{A}^* \) is constructed from \( \mathcal{A} \) by adding the following set of \( \omega \)-edges:

\[
\{(q_1, \gamma, q_2, \text{skip}) \mid s^*(q_1, \gamma, q_2) = \omega\}.
\]

Note that \( s^* \) is a summary function for \( \mathcal{A} \), but not necessarily for \( \mathcal{A}^* \).

**Lemma 51.** For all \( q_1, q_2 \in Q \) and \( \gamma \in \Gamma \), the following assertion holds: the original summary function satisfies \( s(q_1, \gamma, q_2) = \omega \) iff there exists a non-decreasing path in \( \mathcal{A}^* \) from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) that goes through an \( \omega \)-edge.
Proof. The direction from right to left is easy: if there is a non-decreasing path in \( A^* \) that goes through an \( \omega \)-edge, it means that there exists \((q_1', \gamma', q_2')\) with either \( s_d(q_1', \gamma, q_2') = \omega \) or \( s_d(q_1', \gamma', q_2') < s_{d+1}(q_1', \gamma', q_2') \). If \( s_d(q_1', \gamma, q_2') = \omega \), then clearly \( s(q_1, \gamma, q_2) = \omega \). Otherwise we have \( s_d(q_1', \gamma', q_2') < s_{d+1}(q_1', \gamma', q_2') \), and then by Lemma 50 we get that \( s(q_1', \gamma', q_2') = \omega \). Since there exists a finite path from \((\bot, \gamma, q_1)\) to \((\bot, \gamma, q_2)\) with an \( \omega \)-edge it follows that \( s(q_1, \gamma, q_2) = \omega \).

For the converse direction, we consider the case that \( s(q_1, \gamma, q_2) = \omega \). If \( s^*(q_1, \gamma, q_2) = \omega \), then the proof follows immediately. Otherwise it follows that \( s_d(q_1, \gamma, q_2) \in \mathbb{Z} \). Hence there exists a weight \( n \in \mathbb{Z} \) such that a non-decreasing path with minimal additional stack height with weight \( n \) has additional stack height \( d' \geq d+1 \). Let \( \pi \) be such a path. Then there exists a non-decreasing subpath that starts at \((\alpha \gamma', q_1')\) and ends at \((\alpha \gamma', q_2')\) with additional stack height exactly \( d + 1 \) (for some states \( q_1', q_2' \) and stack symbol \( \gamma' \)). If \( s_{d+1}(q_1', \gamma', q_2') = s_d(q_1', \gamma', q_2') \), then \( \pi \) is not a path with the minimal additional stack height. Hence, as \( s_{d+1}(q_1', \gamma', q_2') > s_d(q_1', \gamma', q_2') \), by definition \( s^*(q_1', \gamma', q_2') = \omega \) and the proof follows.

\[ \Box \]

We are now ready to show that the summary function \( s \) can be computed polynomial time.

Remark 1. We show that the number of arithmetic operations required is polynomial in the size of the WPS, and hence the polynomial time bound follows. In the sequel, instead of polynomial number of operations in the size of the WPS we simply write polynomial time.

Lemma 52. For a WPS \( A \), the summary function \( s \) is computable in polynomial time.

Proof. There are two key steps of the proof: (i) computation of \( s_d \), for \( d = (|Q| \cdot |\Gamma|)^2 \), and we will argue how to compute \( s_{i+1} \) from \( s_i \) in polynomial time; and (ii) computation of a non-decreasing path in \( A^* \) that goes through an \( \omega \)-edge. We first argue how the key steps give us the desired result and then present the details of the key steps. Given the computation of (i), we construct \( s_d, s_{d+1} \) in polynomial time, and hence also \( s^* \). Given \( s^* \) we construct \( A^* \) in polynomial time. By computation (ii) we can assign the \( \omega \) values for the summary function, and all other have values as defined by \( s_d \). Thus with the computation of key steps (i) and (ii) in polynomial time, we can compute the summary function \( s \) in polynomial time. We now describe the key steps:

1. Computation of \( s_{i+1} \) from \( s_i \) and \( A \). Let \( G_A \) be the finite weighted graph that is formed by all the configurations of \( A \) with stack height either one or two, that is, the vertices are of the form \((\alpha, q)\) where \( q \in Q \) and \( \alpha \in \{\bot \cdot \gamma, \bot \cdot \gamma_1 \cdot \gamma_2 \mid \gamma, \gamma_1, \gamma_2 \in \Gamma\} \). The edges (and their weights) are according to the transitions of \( A \) formally, (i) (Skip edges): for vertices \((\bot \cdot \alpha, q)\) we have an edge to \((\bot \cdot \alpha, q')\) iff \( e = (q, \text{Top}(\alpha), \text{skip}, q') \) is an edge in \( A \) (and the weight of the edge in \( G_A \) is \( w(e) \)) where \( \alpha = \gamma \) or \( \alpha = \gamma_1 \cdot \gamma_2 \) for \( \gamma, \gamma_1, \gamma_2 \in \Gamma \); (ii) (Push edges): for vertices
\((\bot \cdot \gamma, q)\) we have an edge to \((\bot \cdot \gamma \cdot \gamma', q')\) iff \(e = (q, \gamma, push(\gamma'), q')\) is an edge in \(A\) (and the weight of the edge in \(G_A\) is \(w(e)\)) for \(\gamma, \gamma' \in \Gamma\); and (iii) (Pop edges): for vertices \((\bot \cdot \gamma \cdot \gamma', q)\) we have an edge to \((\bot \cdot \gamma, q')\) iff \(e = (q, \gamma', pop, q')\) is an edge in \(A\) (and the weight of the edge in \(G_A\) is \(w(e)\)) for \(\gamma, \gamma' \in \Gamma\). Intuitively, \(G_A\) allows skips, push pop pairs, and only one additional push. Note that \(G_A\) has at most \(2 \cdot |Q| \cdot |\Gamma|^2\) vertices, and can be constructed in polynomial time.

For every \(i \geq 1\), given the function \(s_i\), the graph \(G_A^i\) is constructed from \(G_A\) as follows: adding edges \(((\bot \gamma_1 \gamma_2, q_1), (\bot \gamma_1 \gamma_2, q_2))\) (if the edge does not exist already) and changing its weight to \(s_i(q_1, \gamma_2, q_2)\) for every \(\gamma_1, \gamma_2 \in \Gamma\) and \(q_1, q_2 \in Q\). The value of \(s_{i+1}(q_1, \gamma, q_2)\) is exactly the weight of a maximal-weight path between \((\bot \gamma, q_1)\) and \((\bot \gamma, q_2)\) in \(G_A^i\) (with the following convention: \(-\infty < z < \omega, z + \omega = \omega\) and \(z + -\infty = \omega + -\infty = -\infty\) for every \(z \in \mathbb{Z}\)). If in \(G_A^i\) there is a path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) that contains a cycle with positive weight, then we set \(s_{i+1}(q_1, \gamma, q_2) = \omega\). Hence, given \(s_i\) and \(A\), the construction of \(G_A^i\) is achieved in polynomial time, and the computation of \(s_{i+1}\) is achieved using the Bellman-Ford algorithm [41] in polynomial time (a maximal-weight path is a shortest-weight path if we define the edge length as the negative of the edge weight). Also note that the Bellman-Ford algorithm reports cycles with positive weight (that is, negative length) which is required to set \(\omega\) values of \(s_{i+1}\). It follows that we can compute \(s_{i+1}\) given \(s_i\) and \(A\) in polynomial time.

2. \textit{Non-decreasing \(\omega\)-edge path in \(A^*\)}. We reduce the problem of checking if there exists a non-decreasing path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) in \(A^*\) that goes through an \(\omega\)-edge to the problem of pushdown reachability in pushdown systems (or pushdown graphs), which is known to be in \(\text{PTIME} [6, 109]\). The reduction is as follows: for every state \(q \in Q\) we add a fresh (new) state \(q^*\), add a transition (or edge) \((q^*_1, \gamma, q^*_2, \text{com})\) for every \((q_1, \gamma, q_2, \text{com}) \in \Delta\) (i.e., the freshly added states follow the transition in the fresh copy as in the original WPS), and a transition \((q_1, \gamma, q^*_2, \text{com})\) for every transition \((q_1, \gamma, q_2, \text{com})\) that has an \(\omega\) weight (i.e., there is a transition to the fresh copy only for an \(\omega\)-edge). It follows that there exists an \(\omega\)-edge non-decreasing path in \(A^*\) from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) iff the configuration \((\bot \gamma, q^*_2)\) is pushdown reachable from the configuration \((\bot \gamma, q_1)\). Hence it follows that the existence of a non-decreasing \(\omega\)-edge path in \(A^*\) can be determined in polynomial time.

The desired result follows.

Given the computation of the summary function, we will construct a summary graph, and show the equivalence of the existence of good cycles in a WPS with the existence of positive cycles in the summary graph.

\textit{Summary graph and positive simple cycles}. Given a WPS \(A = \langle Q, \Gamma, q_0 \in Q, E \subseteq (Q \times \Gamma) \times (Q \times \text{Com}(\Gamma)), w : E \rightarrow \mathbb{Z}\rangle\) and the summary function \(s\), we construct the
summary graph $\text{Gr}(\mathcal{A}) = (\mathcal{V}, \mathcal{E})$ of $\mathcal{A}$ with a weight function $\overline{w} : \mathcal{E} \to \mathbb{Z} \cup \{\omega\}$ as follows: (i) $\mathcal{V} = Q \times \Gamma$; and (ii) $\mathcal{E} = E_{\text{skip}} \cup E_{\text{push}}$ where $E_{\text{skip}} = \{((q_1, \gamma), (q_2, \gamma)) \mid s(q_1, \gamma, q_2) > -\infty\}$, and $E_{\text{push}} = \{((q_1, \gamma_1), (q_2, \gamma_2)) \mid (q_1, \gamma_1, q_2, \text{push}(\gamma_2)) \in E\}$; and (iii) for all $e = ((q_1, \gamma), (q_2, \gamma)) \in E_{\text{push}}$ we have $\overline{w}(e) = s(q_1, \gamma, q_2)$, and for all $e \in E_{\text{push}}$ we have $\overline{w}(e) = w(e)$ (i.e., according to the weight function of $\mathcal{A}$). A simple cycle $C$ in $\text{Gr}(\mathcal{A})$ is a positive simple cycle iff one of the following conditions holds: (i) either $C$ contains an $\omega$-edge (i.e., edge labeled $\omega$ by $\overline{w}$); or (ii) the sum of the weights of the edges of the cycle $C$ according to $\overline{w}$ is positive.

**Lemma 53.** A WPS $\mathcal{A}$ has a good cycle iff the summary graph $\text{Gr}(\mathcal{A})$ has a positive simple cycle.

**Proof.** If $\mathcal{A}$ has a good cycle, then let $\pi$ be a good cycle. The good cycle $\pi$ is a non-decreasing path $\langle c_1, \ldots, c_n \rangle$ such that $c_1 = (\alpha_1, \gamma, q)$ and either $c_n = (\alpha_1 \gamma \alpha_2, q)$ or $c_n = (\alpha_1 \gamma, q)$ and $w(\pi) > 0$. Let $m_1, \ldots, m_r$ be the local minima along the path. Note that for every $i < r$, either $m_i$ and $m_{i+1}$ have the same stack height or $m_{i+1}$ is reachable from $m_i$ via one push transition. For configuration $c = (\alpha \gamma, q)$, let us denote $\text{Top}(c) = (\gamma, q)$. Hence the path $\text{Top}(m_1), \ldots, \text{Top}(m_r)$ is a cycle in $\text{Gr}(\mathcal{A})$. If the cycle contains an $\omega$-edge, then it is a positive cycle (by the definition of positive cycles in $\text{Gr}(\mathcal{A})$). Otherwise, the weight of the cycle in $\text{Gr}(\mathcal{A})$ is at least $w(\pi)$, and therefore $\text{Gr}(\mathcal{A})$ has a positive cycle (and therefore a positive simple cycle).

The other direction is as follows. Consider a positive cycle in $\text{Gr}(\mathcal{A})$. If the cycle does not contain an $\omega$-edge, then there exists a non-decreasing path in $\mathcal{A}$ with the same weight that forms a good cycle. Otherwise, let $(\gamma, q)$ be a vertex in the cycle, and $((\gamma_1, q_1), (\gamma_1, q_2))$ be an $\omega$-edge in the cycle of $\text{Gr}(\mathcal{A})$. From the construction of $\text{Gr}(\mathcal{A})$, it follows that there exist $\alpha_1, \alpha_2, \alpha_3$ in $\mathcal{A}$ such that the following non-decreasing paths exist:

- A non-decreasing path $\pi_1$ from $(\alpha_1, \gamma, q)$ to $(\alpha_1 \gamma \alpha_2 \gamma_1, q_1)$ (due to the path of the cycle).
- For every $m \in \mathbb{N}$: a non-decreasing path $\pi^m$ from $(\alpha_1 \gamma \alpha_2 \gamma_1, q_1)$ to $(\alpha_1 \gamma \alpha_2 \gamma_1, q_2)$ with weight at least $m$ (due to the $\omega$-edge).
- A non-decreasing path $\pi_2$ from $(\alpha_1 \gamma \alpha_2 \gamma_1, q_2)$ to $(\alpha_1 \gamma \alpha_2 \gamma_1 \alpha_3 \gamma, q)$ (due to the path of the cycle).

Hence, for $m = W \cdot (|\pi_1| + |\pi_2|) + 1$, we get that the path $\pi_1 \pi^m \pi_2$ is a good cycle. This completes both directions of the proof and gives us the result.

Since the summary function and the summary graph can be constructed in polynomial time, and the existence of a positive cycle in a graph can be checked in polynomial time (for example, first checking the existence of a cycle with an $\omega$-edge, and then applying Karp’s mean-cycle algorithm [61] after removing all $\omega$ edges), we have the following lemma.
Lemma 54. Given a WPS $A$, whether $A$ has a good cycle can be decided in polynomial time.

Lemma 49 and Lemma 54 give us the following theorem.

Theorem 11. Given a WPS $A$, whether there exists an infinite path $\pi$ such that $\text{LimInfAvg}(\pi) > 0$ (or $\text{LimSupAvg}(\pi) > 0$) can be decided in polynomial time. If there exists an infinite path $\pi$ such that $\text{LimSupAvg}(\pi) > 0$, then there exists an ultimately periodic infinite path $\pi'$ such that both $\text{LimSupAvg}(\pi') > 0$ and $\text{LimInfAvg}(\pi') > 0$.

7.1.2 Objectives $\text{LimInfAvg} \geq 0$ and $\text{LimSupAvg} \geq 0$

In this section we consider mean-payoff objectives with non-strict inequality. We will assume that the input WPS $A$ has integer weights, but we will consider certain transformations that produce rational weight functions. We note that we can easily transform rational weights back to integer weights by multiplying all the weights by the least common multiple (LCM) of all the denominators of the weights. As a result the mean-payoff value of every path is multiplied by the least common multiple, but since we only ask if the mean-payoff value is positive (or non-negative), the result does not change. We also note that the least common multiple is bounded by $D \cdot |E|$, where $D$ is the greatest denominator that occurs in the weight function (in absolute value) and $|E|$ is the number of transitions. Hence, the least common multiple requires only $|E| \cdot \log(D)$ bits to encode and the blowup is polynomial.

Transformed weight functions and weighted graphs. Let $w : E \rightarrow \mathbb{Q}$ be a weight function, and $r \in \mathbb{Q}$ be a rational value, then the weight function $w + r : E \rightarrow \mathbb{Q}$ is defined as follows: for all $e \in E$ we have $(w + r)(e) = w(e) + r$. Let $G = (V, E)$ be a (possibly infinite) graph with a weight function $w : E \rightarrow \mathbb{Q}$. In order to emphasize that $w$ is the weight function for $G$, we use $w_G$. We denote by $G^r$ the same infinite graph with weight function $w_G + r$. We first show that if the lim-inf-average objective can be satisfied for all $\epsilon > 0$, then the non-strict lim-inf-average objective can also be satisfied.

Lemma 55. Let $A$ be a WPS. There exists a path $\pi$ with $\text{LimInfAvg}(\pi) \geq 0$ iff for every $\epsilon > 0$ there exists a path $\pi_\epsilon$ with $\text{LimInfAvg}(\pi_\epsilon) > -\epsilon$.

Proof. The direction from left to right is trivial. In order to prove the converse direction let us assume that for every $n \in \mathbb{N}$ there exists a path $\pi_n$ with $\text{LimInfAvg}(\pi_n) > -\frac{1}{n}$. Hence for every $n \in \mathbb{N}$ there exists a path $\pi_n^*$ which leads to a path $C_n$ that is a good cycle with respect to the weight function $w + \frac{1}{n}$. Since there are infinitely many values of $n \in \mathbb{N}$, and since $Q$ and $\Gamma$ are finite, w.l.o.g all the good cycles (with respect to $w + \frac{1}{n}$) start at the same top configuration $(\gamma, q)$. We define an infinite path $\pi = \pi_1^*C_1^{w:|C_2|}C_2^{w:|C_3|} \cdots C_i^{w:|C_{i+1}|} \cdots$ such that $|C_{i+1}| > |C_i|$ (since we can always extend the length of a cyclic path by taking several copies of it). We claim that $\text{LimInfAvg}(\pi) \geq 0$. To prove the claim it is enough to show that for all $\nu > 0$ we have

\footnote{In this subsection we often look at a WPS as an infinite graph of the configurations.}
\[ \text{LimInfAvg}(\pi) \geq -\nu, \] and for this purpose it is enough to prove that for the suffix \( \pi' \) that begins at position \( |\pi| \) we have \( \text{LimInfAvg}(\pi') \geq -\nu. \) For every \( \ell \in \mathbb{N} \) we denote by \( \pi'(\ell) \) the prefix of \( \pi' \) that ends at position \( \sum_{i=1}^{\ell} i \cdot |C_i| \cdot |C_{i+1}| \cdot |W|. \) Since \( C_i \) is a good cycle for \( w + \frac{1}{\ell} \) we get that \( w(C_i) \geq -\frac{\ell}{|W|} \), hence the average weight of \( \pi'(\ell) \) is at least
\[
\frac{\sum_{i=1}^{\ell} -i \cdot W \cdot |C_i| \cdot |C_{i+1}|}{\sum_{i=1}^{\ell} i \cdot |C_i| \cdot |C_{i+1}| \cdot W} = -\frac{\sum_{i=1}^{\ell} |C_i| \cdot |C_{i+1}| \cdot |W|}{\sum_{i=1}^{\ell} i \cdot |C_i| \cdot |C_{i+1}|}.
\]

Since \( |C_{i+1}| > |C_i| \), we get that
\[
\sum_{i=1}^{\ell} |C_i| \cdot |C_{i+1}| \leq 2 \cdot \sum_{i=\ell}^{\ell} |C_i| \cdot |C_{i+1}|; \quad \text{and} \quad \sum_{i=1}^{\ell} i \cdot |C_i| \cdot |C_{i+1}| \geq \frac{\ell}{2} \cdot \sum_{i=\ell}^{\ell} |C_i| \cdot |C_{i+1}|,
\]

and thus we get that the average weight of \( \pi'(\ell) \) is at least \(-\frac{4}{\ell}\). Since each copy of \( C_{\ell+1} \) has an average weight of at least \(-\frac{1}{\ell^2}\) it is obvious that the average value of \( \pi'(\ell) C_{\ell+1} \) remains at least \(-\frac{4}{\ell}\). Moreover, since \( |\pi'(\ell)| \geq \ell \cdot W \cdot |C_{\ell+1}| \) we get that the average value of \( \pi'(\ell) C_{\ell+1} D \) (where \( D \) is a prefix of the finite path \( C_{\ell+1} \)) is at least
\[
-\frac{4 \cdot \ell \cdot W \cdot |C_{\ell+1}| - W \cdot |D|}{\ell \cdot W \cdot |C_{\ell+1}| + |D|} \geq \frac{5}{\ell}.
\]

Hence, we get that for all \( \nu > 0 \), every prefix of \( \pi' \) that is longer than \( |\pi'|(|\frac{2}{\ell^2})| \) has an average weight of at least \(-\nu\) and thus \( \text{LimInfAvg}(\pi) = \text{LimInfAvg}(\pi') \geq -\nu \) and the proof is completed.

\[ \square \]

**Lemma 56.** Let \( A \) be a WPS with integer weights (weight function \( w \)). Let \( \ell = |\Gamma| \cdot |Q| \), and fix \( \varepsilon = \frac{1}{\ell + \frac{1}{2} \ell^2}. \) Then the WPS \( A^\varepsilon \) (with weight function \( w + \varepsilon \)) has a good cycle iff for every \( \delta > 0 \) the WPS \( A^\delta \) (with weight function \( w + \delta \)) has a good cycle. Moreover, every good cycle in \( \text{Gr}(A^\varepsilon) \) is a good cycle in \( \text{Gr}(A^\delta) \).

**Proof.** The direction from right to left is trivial. For the converse direction we first prove the following lemma.

**Lemma 57.** Let \( s^\varepsilon \) be the summary function for \( A^\varepsilon \).

1. If \( s^\varepsilon(q_1, \gamma, q_2) \neq \omega \), then \( s^\varepsilon(q_1, \gamma, q_2) \leq s(q_1, \gamma, q_2) + \frac{1}{\ell^2}. \)

2. If \( s^\varepsilon(q_1, \gamma, q_2) = \omega \), then for every \( \delta > 0 \) we have \( s^\delta(q_1, \gamma, q_2) = \omega \), where \( s^\delta \) is the summary function for \( A^\delta \).

**Proof.** We prove both the items below.

1. If \( s^\varepsilon(q_1, \gamma, q_2) \neq \omega \), then consider a maximal-weight non-decreasing path with minimal additional stack height from \( (\bot, \gamma, q_1) \) to \( (\bot, \gamma, q_2) \) that has an additional stack height of at most \((|Q| \cdot |\Gamma|)^2 = \ell^2 \). Note that this path does not contain
positive cycles (since \( s^*(q_1, \gamma, q_2) \neq \omega \)). Hence there exists a path \( \pi \) with the same weight and with stack height at most \( \ell^2 \) which does not contain any cycles. Hence \( \|\pi\| \leq \ell^2 \), and therefore

\[
\omega_A^*(\pi) = \omega_A(\pi) + \epsilon \cdot \|\pi\| \leq \omega_A(\pi) + \epsilon \cdot \ell^2 \leq \omega_A(\pi) + \frac{1}{2 \cdot \ell}.
\]

Since \( s(q_1, \gamma, q_2) \geq \omega_A(\pi) \) (as \( \pi \) is a non-decreasing path we have \( s(q_1, \gamma, q_2) \geq \omega_A(\pi) \)), we obtain the result of the first item.

2. In order to prove the second item of the lemma, it is enough to prove that if an edge weight is \( \omega \) in \( (A^\epsilon)^* \) (where \( (A^\epsilon)^* \) is the WPS constructed with the function \( (s^\epsilon)^* \)), then for every \( \delta > 0 \) the weight of the edge is also \( \omega \) in the summary graph \( Gr(A^\delta) \) of \( A^\delta \). We consider two cases to complete the proof.

- **Case 1.** If \( s^\epsilon_A(q_1, \gamma, q_2) = \omega \), then the infinite graph \( A^\epsilon \) has a positive cycle \( C \) with stack height at most \( \ell^2 \), and hence there exists a positive cycle \( C' \) such that \( |C'| \leq \ell^2 \). Towards a contradiction, let us assume that \( \omega_A(C') < 0 \). As all the weights in \( A \) are integers we get that \( \omega_A(C') \leq -1 \). As \( \omega_A(C') + \epsilon \cdot |C'| = \omega_A(C') \geq 0 \) we get that \( |C'| \geq \frac{1}{\epsilon} \), which is a contradiction. Thus \( \omega_A(C') \geq 0 \), and hence for every \( \delta > 0 \) we have \( \omega_A^*(C') > 0 \). Thus \( s^\delta_A(q_1, \gamma, q_2) = \omega \).

- **Case 2.** Otherwise, we have \( s^\epsilon_A(q_1, \gamma, q_2) > s^\epsilon_A(q_1, \gamma, q_2) \). Let \( \pi \) be a path from \((\perp, \gamma, q_1)\) to \((\perp, \gamma, q_2)\) with additional stack height \( \ell + 1 \) and weight \( s^\epsilon_A(q_1, \gamma, q_2) \). As \( s^\epsilon_A(q_1, \gamma, q_2) > s^\epsilon_A(q_1, \gamma, q_2) \), by Lemma 45 it follows that \( \pi \) has a pumpable pair \((p_1, p_2)\) with \( \omega_A(p_1) + \omega_A(p_2) > 0 \). If \( p_1 \) (resp. \( p_2 \)) contains a positive cycle, then by the same arguments presented in the proof of the first item of the lemma this cycle will be positive also in \( A^\delta \), for every \( \delta > 0 \), and hence \( s^\delta_A(q_1, \gamma, q_2) = \omega \). If \( p_1 \) (resp. \( p_2 \)) contains a non-negative cycle, then we can remove the cycle and still obtain a pumpable pair with sum of weights positive. Therefore w.l.o.g both \( p_1 \) and \( p_2 \) do not contain any cycles and thus \( |p_1|, |p_2| \leq \ell^\delta + 1 \). Again by the same arguments presented in the proof of the first item we obtain that \( \omega_A(p_1) + \omega_A(p_2) \geq 0 \) and hence for every \( \delta > 0 \) we have \( \omega_A(p_1) + \omega_A(p_2) > 0 \). As \((p_1, p_2)\) is a positive pumpable pair in \( A^\delta \) it follows that \( s^\delta_A(q_1, \gamma, q_2) = \omega \).

This completes the proof of the second item.

We obtain the desired result of the lemma.

\( \square \)

We are now ready to prove Lemma 56. Let us assume that there exists a good cycle in \( A^\epsilon \). Then by Lemma 53 there exists a positive simple cycle \( C \) in the summary graph \( Gr(A^\epsilon) \). We consider two cases:
• If $C$ contains an $\omega$-edge $e$, then by Lemma 57 for every $\delta > 0$ the same cycle in $Gr(A^{\delta})$ will also contain an $\omega$-edge. Therefore $C$ is a positive cycle also in $Gr(A^{\delta})$ and hence $A^{\delta}$ has a good cycle.

• Otherwise $C$ does not contain an $\omega$-edge. Towards a contradiction assume that the weight of $C$ in $Gr(A^{\epsilon})$ is negative. As the weights of $A$ are integers it follows that the weight of $C$ is at most $-1$. By Lemma 57, for every $e \in C$ we have $w_{Gr(A^{\epsilon})}(e) \leq w_{Gr(A)}(e) + \frac{|C|}{2}$, and thus $w_{Gr(A^{\epsilon})}(C) \leq w_{Gr(A)}(C) + \frac{|C|}{2} \leq -\frac{1}{2}$, which contradicts the assumption that $C$ is a positive cycle. Therefore we have $w_{Gr(A)}(C) \geq 0$, and therefore for every $\delta > 0$ we get that $w_{Gr(A^{\delta})}(C) > 0$ and hence $A^{\delta}$ has a good cycle.

The moreover part of the lemma follows from the fact that $C$ is an arbitrary positive cycle in $Gr(A^{\epsilon})$. This completes the proof of the lemma.

Theorem 12. Given a WPS $A$, whether there exists an infinite path $\pi$ such that $\text{LimInfAvg}(\pi) \geq 0$ (or $\text{LimSupAvg}(\pi) \geq 0$) can be decided in polynomial time. There exists a WPS $A$ such that there exists a path $\pi$ with $\text{LimInfAvg}(\pi) = 0$ but for every ultimately periodic path $\pi$ we have both $\text{LimInfAvg}(\pi) < 0$ and $\text{LimSupAvg}(\pi) < 0$.

Proof. From Lemma 55 it follows that if there is a path $\pi$ such that $\text{LimInfAvg}(\pi) \geq 0$, then for every $\epsilon_1 > 0$ there is a path $\pi'$ such that $\text{LimInfAvg}(\pi') > -\epsilon_1$. By Lemma 56 it follows that it suffices to check for $\epsilon$ (for the $\epsilon$ described by Lemma 56). Given a WPS $A$, the WPS $A^\epsilon$ can be constructed in polynomial time (as $\epsilon$ only has polynomial number of bits). Then applying the polynomial-time algorithm to find good cycles (as given in the previous subsection) we answer the decision problems in polynomial time. We observe that Lemma 55 and Lemma 56 also hold for $\text{LimSupAvg}$ objectives, and thus the result also follows for $\text{LimSupAvg}$ objectives.

Example 6 shows that the witness paths for non-strict inequality mean-payoff objectives are not necessarily ultimately periodic.

We get the next corollary from Lemma 56.

Corollary 1. Given a WPS $A$ with integer weights, let $\epsilon$ be the constant from Lemma 56. Then there is a path $\pi$ with $\text{LimInfAvg}(\pi) > 0$ if and only there is a path $\pi$ with $\text{LimInfAvg}(\pi) \geq \epsilon$.

Proof. Towards a contradiction we assume that there is a path $\pi$ such that $\text{LimInfAvg}(\pi) > 0$ but for all paths $\pi$ we have $\text{LimInfAvg}(\pi) < \epsilon$. Consider the WPS $A^{-\epsilon}$ obtained by subtracting $\epsilon$ from all the weights of $A$. Then in $A^{-\epsilon}$ there are no good cycles, that is, for all the cycles $C$ in $Gr(A^{-\epsilon})$ the sum of the weights of the cycle...
$C$ is negative (note that if $\text{Gr}(\mathcal{A}^{-\epsilon})$ has a cycle with non-negative sum of weights then we could obtain a path $\pi$ with $\text{LimInfAvg}(\pi) \geq \epsilon$). We construct a WPS $(-\mathcal{A})^\epsilon$ by multiplying all the weights of $\mathcal{A}^{-\epsilon}$ by $-1$. Since all the cycles in $\text{Gr}(\mathcal{A}^{-\epsilon})$ have negative sum of weights it follows that all the cycles in $\text{Gr}((-\mathcal{A})^\epsilon)$ are good. Hence, by Lemma 56 it follows that for every $\delta > 0$, all the cycles in $\text{Gr}((-\mathcal{A})^\delta)$ are good. Therefore, for every $\delta > 0$ we get that all the cycles in $\text{Gr}(\mathcal{A}^{-\delta})$ have negative sum of weights and it follows that for all paths $\pi$ in $\mathcal{A}^{-\delta}$ we have $\text{LimInfAvg}(\pi) < 0$. Thus, we conclude that for every $\delta > 0$ and for every infinite path $\pi$ we have $\text{LimInfAvg}(\pi) \leq \delta$, which contradicts the assumption that there exists a path $\pi$ with $\text{LimInfAvg}(\pi) > 0$ (since surely, for some $\delta > 0$ it must hold that $\text{LimInfAvg}(\pi) > \delta$).

\[\square\]

**Corollary 2.** Given a WPS $\mathcal{A}$, whether there exists an ultimately periodic infinite path $\pi$ such that $\text{LimInfAvg}(\pi) \geq 0$ (or $\text{LimSupAvg}(\pi) \geq 0$) can be decided in polynomial time.

**Proof.** We first observe that for any ultimately periodic path $\pi$ we have $\text{LimInfAvg}(\pi) = \text{LimSupAvg}(\pi)$. Hence, it is enough to prove the assertion for the $\text{LimInfAvg}(\pi)$ $\geq 0$ objective. The proof will immediately follow from the next claim: There exists an ultimately periodic path $\pi = \pi_0(\pi_1)^\omega$ with $\text{LimInfAvg}(\pi) \geq 0$ if and only if the summary graph $\text{Gr}(\mathcal{A})$ contains a (reachable) cycle with non-negative sum of weights. We first prove the claim. The proof for the direction from right to left is straightforward and is as follows. If $\text{Gr}(\mathcal{A})$ has a cycle with non-negative sum of weights, then there exists a non-decreasing cyclic path $\pi_1$ that begins in configuration $(\alpha \gamma, q)$ with $w(\pi_1) \geq 0$. Since the cycle is reachable, then there is a path $\pi_0$ that begins in $(\perp, q_0)$ and ends in $(\alpha \gamma, q)$. Since $\pi_1$ is non-decreasing, the path $\pi_0(\pi_1)^\omega$ is a valid infinite path, and since $w(\pi_1) \geq 0$ we get that $\text{LimInfAvg}(\pi_0(\pi_1)^\omega) = \lim_{\pi_0} \frac{w(\pi_1)}{\pi_1} \geq 0$. To prove the converse direction, we assume that all the cycles in $\text{Gr}(\mathcal{A})$ have negative sum of weights and show that for all ultimately periodic paths $\pi$ we have $\text{LimInfAvg}(\pi) < 0$. Let $\pi = \pi_0(\pi_1)^\omega$ be a valid path in $\mathcal{A}$, and let $c_1, c_2, \ldots$ be the configurations of $\pi$ and $m_1, m_2, \ldots$ be the sequence of infinitely many local minima in $\pi$. Let $(\gamma, q)$ be a stack symbol and a state such that $|\{i \mid \text{Top}(m_i) = (\gamma, q)\}| = \infty$. Since $\pi$ is ultimately periodic, then for some index $i$ we get that for every $j \in \mathbb{N}$ it holds that $\text{Top}(c_{i+j|\pi_1|j}) = (\gamma, q)$. Hence, if we denote by $W(\gamma, q)$ the weight of the maximal-weight non-decreasing path from $(\gamma, q)$ to $(\gamma, q)$ we get that $\text{LimInfAvg}(\pi = \pi_0(\pi_1)^\omega) \leq \frac{W(\gamma, q)}{\pi_1}$. Since all the cycles in $\text{Gr}(\mathcal{A})$ have negative sum of weights we get that $W(\gamma, q) < 0$, and therefore $\text{LimInfAvg}(\pi) < 0$. Hence, the polynomial-time algorithm is to construct $\text{Gr}(\mathcal{A})$ and to detect the existence of a non-negative cycle.

\[\square\]

### 7.1.3 Mean-payoff objectives with stack boundedness

In this section we consider WPSs with mean-payoff objectives along with the stack boundedness condition that requires the height of the stack to be bounded. An infinite
path $\pi = \langle c_1, c_2, \ldots, c_i, \ldots \rangle$ is a stack bounded path if there exists $n \in \mathbb{N}$ such that $|\alpha_i| \leq n$ for every $i \in \mathbb{N}$ (recall that $\alpha_i$ is the stack string of configuration $c_i$).

**Theorem 13.** Given a WPS $A$, the following problems can be solved in PTIME.

1. Does there exist a stack bounded infinite path $\pi$ such that $\text{LimInfAvg}(\pi) \gg 0$ (resp. $\text{LimSupAvg}(\pi) \gg 0$), for $\gg \in \{\geq, >\}$?

2. Is $\sup\{\text{LimInfAvg}(\pi) \mid \pi \text{ is a stack bounded path} \} \geq 0$ (resp. $\sup\{\text{LimSupAvg}(\pi) \mid \pi \text{ is a stack bounded path} \} \geq 0$)?

**Proof.** The results for each item are proved with a lemma below.

**Lemma 58.** There exists a stack bounded infinite path $\pi$ in $A$ such that $\text{LimSupAvg}(\pi) > 0$ (resp. $\text{LimSupAvg}(\pi) \geq 0$) iff the summary graph $\text{Gr}(A)$ has a vertex with self-loop that has a positive (resp. non-negative) weight.

**Proof.** If there exists a stack bounded infinite path $\pi$ in $A$ such that $\text{LimSupAvg}(\pi) > 0$ (resp. $\text{LimSupAvg}(\pi) \geq 0$), then it contains a cycle that begins and ends at configuration $(\alpha\gamma, q)$ with positive (resp. non-negative) weight. Hence in the summary graph $\text{Gr}(A)$ the vertex $(\gamma, q)$ will have a self-loop with positive (resp. non-negative) weight. Conversely, if there is a (reachable) vertex $(\gamma, q)$ in $\text{Gr}(A)$ with a positive (resp. non-negative) weight self-loop, then for some stack string $\alpha$ there is a non-decreasing path $\pi_1$ from $(\alpha\gamma, q)$ to $(\alpha\gamma, q)$ with $w(\pi_1) > 0$ (resp. $w(\pi_1) \geq 0$) and there is a path $\pi_0$ from $(\bot, q_0)$ to $(\alpha\gamma, q)$. Hence the stack height of the path $\pi = \pi_0(\pi_1)^\omega$ is bounded by $\text{ASH}(\pi_0) + \text{ASH}(\pi_1)$, and $\text{LimSupAvg}(\pi) > 0$ (resp. $\text{LimSupAvg}(\pi) \geq 0$).

It is straightforward to verify that Lemma 58 also holds for $\text{LimInfAvg}(\pi)$ objectives (we simply replace every occurrence of $\text{LimSupAvg}$ by $\text{LimInfAvg}$ and the same argument holds). This gives us the first item of the theorem. The next lemma proves the last item of the theorem.

**Lemma 59.** Let $\epsilon$ be the constant from Lemma 56. Then there exists a stack bounded infinite path $\pi$ such that $\text{LimInfAvg}(\pi) > -\epsilon$ iff $\sup\{\text{LimInfAvg}(\pi) \mid \pi \text{ is a stack bounded path} \} \geq 0$.

**Proof.** The direction from right to left is immediate (clearly, if all paths $\pi$ have $\text{LimInfAvg}(\pi) \leq -\epsilon$, then $\sup\{\text{LimInfAvg}(\pi) \mid \pi \text{ is a stack bounded path} \} < 0$). In order to prove the other direction let us assume that there exists a stack bounded infinite path $\pi$ such that $\text{LimInfAvg}(\pi) > -\epsilon$. Hence by Lemma 58 the summary graph $\text{Gr}(A^\epsilon)$ contains a self-loop for vertex $(\gamma, q)$ with positive weight. By the same argument used in the proof of Lemma 56 it follows that for every $\delta > 0$ the self-loop of vertex $(\gamma, q)$ will have a positive weight in graph $\text{Gr}(A^\delta)$. Hence for every $\delta > 0$ there exists a stack bounded path $\pi_\delta$ such that $\text{LimInfAvg}(\pi_\delta) > -\delta$, which implies that $\sup\{\text{LimInfAvg}(\pi) \mid \pi \text{ is a stack bounded path} \} \geq 0$. 

\qed
The proof of Lemma 59 straightforwardly extends to $\text{LimSupAvg}(\pi)$ objectives (we simply replace every occurrence of $\text{LimInfAvg}$ by $\text{LimSupAvg}$ and the same argument holds), and hence we have the desired result of the theorem.

Thus we have the following result summarizing the computational complexity.

**Theorem 14.** Given a WPS $A$, the following questions can be solved in polynomial time: (1) Whether there exists a path $\pi$ in $\Phi \bowtie 0$, where $\Phi \in \{\text{LimSupAvg}, \text{LimInfAvg}\}$ and $\bowtie \in \{\geq, >\}$; and (2) whether there exists a path $\pi$ in $\Phi \bowtie 0$ such that $\pi$ is stack bounded, where $\Phi \in \{\text{LimSupAvg}, \text{LimInfAvg}\}$ and $\bowtie \in \{\geq, >\}$.

### 7.2 Pushdown Graphs with (Conjunctive) Multidimensional Mean-payoff Objectives

In this section we consider pushdown graphs (or pushdown systems) with (conjunctive) multidimensional mean-payoff objectives, and we give an algorithm that determines if there exists a path that satisfies a conjunctive multidimensional objective. The algorithm we propose runs in polynomial time even for arbitrary number of dimensions and for arbitrary weight function. We use the hyperplane separation technique from Chapter 3 to reduce the problem into a one-dimensional pushdown graph, and a polynomial solution for the latter was proved in Section 7.1.

**Key obstacles and overview of the solution.** We first describe the key obstacles for the polynomial time algorithm to solve pushdown graphs with multidimensional mean-payoff objectives (as compared to finite-state graphs and finite-state games). For pushdown graphs we need to overcome the next three main obstacles: (a) The mean-payoff value of a finite-state graph is uniquely determined by the weights of the simple cycles of the graph. However, for pushdown graphs it is also possible to pump special types of acyclic paths. Hence, we first need to characterize the pumpable paths that uniquely determine the possible mean-payoff vectors in a pushdown graph. (b) Lemma 2 (from Chapter 3) does not hold for arbitrary infinite-state graphs and we need to show that it does hold for pushdown graphs. (c) We require an algorithm to decide whether there is a hyperplane such that all the weights of the pumpable paths of a pushdown graph lie below the hyperplane (also for arbitrary dimensions). The overview of our solutions to the above obstacles are as follows: (a) In the first part of the section (until Proposition 3) we present a characterization of the pumpable paths in a pushdown graph. (b) We use Gordan’s Lemma [52] (a special case of Farkas’ Lemma) and in Lemma 64 we prove that Lemma 1 and Lemma 2 (from Chapter 3) hold also for pushdown graphs (Lemma 1 holds for any infinite-state graph). (c) Conceptually, we find the separating hyperplane by constructing a matrix $A$, such that every row in $A$ is a weight vector of a pumpable path, and we solve the linear inequality $\bar{\lambda} \cdot A < \bar{\theta}$. However, in general the matrix $A$ can be of exponential size. Thus we need to use advanced linear-programing technique that solves in polynomial time linear inequalities.
CHAPTER 7. MEAN-PAYOFF PUSHDOWN GAMES 117

\begin{equation*}
\text{push}(γ_1), \text{push}(γ_2), (7, 2) \quad \text{pop}(γ_2), (5, -9) \quad \text{pop}(γ_1), (2, -2)
\end{equation*}

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node[state] (q0) {$q_0$};
  \node[state] (q1) [right of=q0] {$q_1$};
  \node[state] (q2) [right of=q1] {$q_2$};
  \node[state] (q3) [right of=q2] {$q_3$};
  \node[state] (q4) [right of=q3] {$q_4$};
  \node[state] (q5) [right of=q4] {$q_5$};

  \path[->] (q0) edge node {$1, 3$} (q1)
                 edge node {$4, 2$} (q2);
  \path[->] (q1) edge node {$7, 1$} (q2);
  \path[->] (q2) edge node {$2, 6$} (q3);
  \path[->] (q3) edge node {$2, 8$} (q4);
  \path[->] (q4) edge node {$7, 1$} (q5);
  \path[->] (q5) edge node {$2, 6$} (q0);
\end{tikzpicture}
\caption{A WPS $A$. If an edge is not label with a command, then the command is \textit{skip}. The label \textit{pop}(γ) stands for: if the top symbol is γ, then a pop transition is possible.}
\end{figure}

with polynomial number of variables and exponential number of constraints. This technique requires a polynomial-time oracle that for a given $\vec{λ}$ returns a violated constraint (or says that all constraints are satisfied). We show that in our case the required oracle is the algorithm for pushdown graphs with one-dimensional mean-payoff objective (which we obtain from Section 7.1), and thus we establish a polynomial-time hyperplane separation technique for pushdown graphs.

Multi-weighted pushdown systems. A multi-weighted pushdown system (WPS) (or a multi-weighted pushdown graph) is a tuple:

\[ A = \langle Q, Γ, q_0 \in Q, E \subseteq (Q \times Γ) \times (Q \times \text{Com}(Γ)), w : E \to \mathbb{Z}^k \rangle, \]

where $Q$ is a finite set of states with $q_0$ as the initial state; $Γ$ the finite stack alphabet and we assume there is a special initial stack symbol $\perp \in Γ$; $E$ describes the set of edges or transitions of the pushdown system; and $w$ is a weight function that assigns an integer weight vector to every edge; we denote by $w_i$ the projection of $w$ to the $i$-th dimension. We assume that $\perp$ can be neither put on nor removed from the stack.

The basic definitions of configurations, runs, stack height, pumpable pair of paths, local minimum of path etc., are identical to the one-dimensional definitions in Section 7.1. The next definition is new and relevant only to multidimensional weighted systems.

\textit{Cone of pumpable pairs}. We denote $\mathbb{R}_+ = [0, +\infty)$. For a finite non-decreasing path $π$ we denote by $\text{PPS}(π)$ the (finite) set of pumpable pairs that occur in $π$, that is, $\text{PPS}(π) = \{(p_1, p_2) \in (E^* \times E^*) \mid p_1$ and $p_2$ are a pumpable pair in $π\}$. Let $\text{PPS}(π) = \{P_1 = (p_1^1, p_1^2), P_2 = (p_2^1, p_2^2), \ldots, P_j = (p_j^1, p_j^2)\}$, and we denote by $\text{PumpMat}(π)$ the matrix that is formed by the weight vectors of the pumpable pairs of $π$, that is, the matrix has $j$ rows and the $i$-th row of the matrix is $w(p_i^1) + w(p_i^2)$ (every weight vector is a row in the matrix). We denote by $\text{PCon}(π)$ the cone of the weight vectors in $\text{PPS}(π)$, formally, $\text{PCon}(π) = \{\text{PumpMat}(π) \cdot \vec{x} \mid \vec{x} \in (\mathbb{R}_+^k \setminus \{0\})\}$.

\textbf{Example}. We illustrate the definition with the aid of an example. Consider the WPS shown in Figure 7.3. Consider all the possible paths from $(\perp, q_0)$ to $(\perp, q_5)$. Every such
The example illustrates the various concepts we have introduced.

\begin{notations}
Fix $\ell = (|Q| \cdot |\Gamma|)^{|Q| \cdot |\Gamma|}^{2} + 1$ for the rest of the section. For $q_1, q_2 \in Q$ and $\gamma_1, \gamma_2 \in \Gamma$, by abuse of notation we denote by $\text{PPS}(\gamma_1, q_1, \gamma_2, q_2)$ the (finite) set of all pumpable pair of paths, not longer than $\ell$, that occur in a non-decreasing path from $(\gamma_1, q_1)$ to $(\gamma_2, q_2)$; we similarly define $\text{PumpMat}(\gamma_1, q_1, \gamma_2, q_2)$ and $\text{PCon}(\gamma_1, q_1, \gamma_2, q_2)$. If $q_1 = q_2$ and $\gamma_1 = \gamma_2$, then we abbreviate $\text{PPS}(\gamma_1, q_1, \gamma_1, q_1)$ by $\text{PPS}(\gamma_1, q_1)$, and similarly for $\text{PumpMat}$ and $\text{PCon}$.
\end{notations}

In the next lemma we show that any sufficiently long non-decreasing path contains a pumpable pair of paths.

\begin{lemma}
Every non-decreasing path longer than $\ell$ has a pumpable pair of paths.
\end{lemma}

\begin{proof}
Let $\pi$ be a non-decreasing path longer than $\ell$. If $\text{ASH}(\pi) > (|Q| \cdot |\Gamma|)^2$, then by Lemma 44 we get the desired result; otherwise, it is an easy observation that $\pi$ contains a proper cycle, which is by definition a pumpable pair of paths (where one path in the pair is empty).
\end{proof}

\begin{corollary}
Every non-decreasing path longer than $\ell$ has a pumpable pair of paths with length at most $\ell$.
\end{corollary}

The next two lemmas show basic properties of $\text{PPS}$. The first lemma asserts that we can decompose every non-decreasing path to a set of pumpable pairs and a short non-decreasing path.
Lemma 61. For every non-decreasing path \( \pi \) from \((\gamma_1, q_1)\) to \((\gamma_2, q_2)\) there exists a tuple of pumpable pair of paths \( P_1 = (p_1^1, p_1^2), P_2 = (p_2^1, p_2^2), \ldots, P_j = (p_j^1, p_j^2) \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2))\) each of length at most \( \ell \) (i.e., for all \( 1 \leq i \leq j \) we have \(|P_i| \leq \ell\)), a finite non-decreasing path \( \pi_0 \) from \((\gamma_1, q_1)\) to \((\gamma_2, q_2)\) with length at most \( \ell \), and non-negative constants \( m_1, \ldots, m_j \) such that \( w(\pi) = w(\pi_0) + \sum_{i=1}^j m_i \cdot w(P_i) \) and \(|\pi| = |\pi_0| + \sum_{i=1}^j m_i \cdot |P_i|\).

Proof. The proof is by induction of the length of \( \pi \). If \(|\pi| \leq \ell\), then we are done by choosing \( j = 0 \) and \( \pi_0 = \pi \). Otherwise, by Corollary 3, the path has a pumpable pair \( P = (p_1, p_2) \) with length less than \( \ell \) (and hence \( P \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2))\)). Let \( \pi^* \) be the path that is obtained from \( \pi \) by pumping \( P \) zero times (i.e., \( \pi^* \) is obtained by omitting \( P \) from \( \pi \)); clearly \( \pi^* \) is a non-decreasing path from \((\gamma_1, q_1)\) to \((\gamma_2, q_2)\) and shorter than \( \pi \), any by the induction hypothesis we get the desired result. \( \Box \)

The following lemma shows the connection between the average weight of a path and \( \text{PPS} \).

Lemma 62. If \( \text{PCone}((\gamma_1, q_1), (\gamma_2, q_2)) \cap \mathbb{R}^k_+ = \emptyset \), then there exist constants \( \epsilon > 0 \) and \( m \in \mathbb{N} \), such that for every finite non-decreasing path \( \pi \) from \((\gamma_1, q_1)\) to \((\gamma_2, q_2)\), there exists a dimension \( t \) such that \( w_t(\pi) \leq m - \epsilon \cdot |\pi| \).

Proof. In order to define \( \epsilon \), we consider the following linear programming problem with the variables \( x_1, x_2, \ldots \) and \( r \): the objective function is to maximize \( r \) subject to the constraints below

\[
\sum_{z \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2))} x_z \cdot w_t(z) \geq r \quad \text{for } t = 1, \ldots, k \quad (7.1)
\]

\[
\sum_{z \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2))} x_z = 1 \quad (7.2)
\]

\[
x_z \geq 0 \quad \text{for all } z \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2)) \quad (7.3)
\]

Intuitively, the first constraint specifies that there is a convex combination of the weights of the pumpable pairs to ensure at least \( r \) in every dimension; and the following two constraints is to ensure that it is a convex combination. As the domain of the variables is closed and bounded, there exists a maximum value to the linear program, and let \( r^* \) be the maximum value. If \( r^* \geq 0 \), then we get a contradiction to the assumption that \( \text{PCone}((\gamma_1, q_1), (\gamma_2, q_2)) \cap \mathbb{R}^k_+ = \emptyset \). Hence we have \( r^* < 0 \). We define \( m = (\ell+1)\cdot W - r^* \), \( \epsilon = -\frac{r^*}{m} \) and we claim that for every non-decreasing path \( \pi \) from \((\gamma_1, q_1)\) to \((\gamma_2, q_2)\) there is a dimension \( t \) such that \( w_t(\pi) \leq m - \epsilon \cdot |\pi| \).

By Lemma 61, there exists a path \( \pi_0 \) with length at most \( \ell \), a (finite) sequence of pumpable pairs \( P_1, \ldots, P_j \in \text{PPS}((\gamma_1, q_1), (\gamma_2, q_2)) \) each of length at most \( \ell \) and constants \( m_1, \ldots, m_j \) such that \( w(\pi) = w(\pi_0) + \sum_{i=1}^j m_i \cdot w(P_i) \) and \(|\pi| = |\pi_0| + \sum_{i=1}^j m_i \cdot |P_i|\). We define \( M = \sum_{i=1}^j m_i \). As all \(|P_i|\) and \(|\pi_0|\) are bounded by \( \ell \), we
get that $M \geq \frac{|\pi| - \ell}{\ell}$. Observe that if we set $x_i = \frac{w_i}{M^j}$ for $i = 1$ to $j$, and let $x_z = 0$ for all other $z \in PPS((\gamma_1, q_1), (\gamma_2, q_2))$, then they satisfy the constraints for convex combination. Hence there must exist a dimension $t$ for which $\frac{1}{M} \sum_{i=1}^{j} m_i \cdot w_i(P_i) \leq r^*$ (since $r^*$ is the maximum among the feasible solutions). Thus $w_t(\pi) \leq w_t(\pi_0) + M \cdot r^*$ and since $r^* < 0$ we have

$$w_t(\pi) \leq w_t(\pi_0) + \frac{|\pi| - \ell}{\ell} \cdot r^* = w_t(\pi_0) - r^* + \frac{|\pi| \cdot r^*}{\ell}.$$  

Therefore, for the choice of $m \geq \ell \cdot W - r^*$ and $\epsilon = -\frac{r^*}{\ell}$, we obtain the desired result. \(\square\)

The next proposition gives a sufficient and necessary condition for the existence of a path with non-negative mean-payoff values in all the dimensions.

**Proposition 3.** There exists an infinite path $\pi$ such that $\text{LimInfAvg}(\pi) \geq \bar{0}$ if and only if there exists a (reachable) non-decreasing cycle $\pi$ such that $\mathbb{R}^k_+ \cap \text{PCone}(\pi) \neq \emptyset$.

**Proof.** We first prove the direction from right to left. If there exists a path $\pi$ such that $\mathbb{R}^k_+ \cap \text{PCone}(\pi) \neq \emptyset$, then by definition there are $j$ pumpable pairs $P_1, P_2, \ldots, P_j$ with weight vectors $y_1 = w(P_1), \ldots, y_j = w(P_j)$ such that there exist $j$ positive constants (w.l.o.g natural positive constants) $n_1, \ldots, n_j$ such that $\sum_{i=1}^{j} n_i \cdot y_i \geq \bar{0}$.

For every $a, b \in \mathbb{N}$ we denote by $\pi^{a,b}$ the (finite) path that is formed by pumping the $a$-th pumpable pair $b$ times. We denote by $\pi^b = \pi^{1, b} \cdot \pi^{2, b} \cdot \pi^{3, b} \cdot \cdots$, where the $i$-th pumpable pair is pumped $b \cdot n_i$ times, respectively. We note that $\pi^{a,b}$ is a non-decreasing cycle, and for the infinite path

$$\pi^* = \pi^1 \cdot \pi^2 \cdot \pi^3 \cdot \cdots$$

we get $\text{LimAvg}(\pi^*) \geq \bar{0}$. The reason we have $\text{LimAvg}(\pi^*) \geq \bar{0}$ as $b$ tends to infinity, the average weight is determined only by the weights of the $j$ pumpable pairs and their coefficients $n_1, \ldots, n_j$, and we have $\sum_{i=1}^{j} n_i \cdot y_i \geq \bar{0}$. This completes the proof for the direction from right to left.

For the converse direction, let $\pi$ be an infinite path such that $\text{LimAvg}(\pi) \geq \bar{0}$, and let $(\gamma, q)$ be a top configuration that occurs infinitely often in the local minimum of $\pi$. Since $\text{LimAvg}(\pi) \geq \bar{0}$ it follows that for every $\epsilon > 0$ there exists a non-decreasing cycle that begins at $(\gamma, q)$ with average weight at least $-\epsilon$ in every dimension. Hence, by Lemma 62 it follows that $\text{PCone}(\gamma, q) \cap \mathbb{R}^k_+ \neq \emptyset$, and hence, there exists a non-decreasing cycle $\pi$ that starts in $(\gamma, q)$ for which $\text{PCone}(\pi) \cap \mathbb{R}^k_+ \neq \emptyset$. \(\square\)

By Proposition 3, we can decide whether there is an infinite path $\pi$ for which $\text{LimAvg}(\pi) \geq \bar{0}$ by checking if there exist a tuple $(\gamma, q) \in \Gamma \times Q$ for which there is a non-negative (and non-trivial) solution for the equation $\text{PumpMat}((\gamma, q)) \cdot \bar{x} \geq 0$. As in Lemma 1 by adding $k$ self-loop transitions with weights, where the weight of transition $i$ is $-1$ in the $i$-th dimension and $0$ in the other dimensions, we reduce the problem to finding $q$ and $\gamma$ such that there is a non-negative solution for $\text{PumpMat}((\gamma, q)) \cdot \bar{x} = 0$. Inspired by the techniques of [40], we present an algorithm that solves the problem by a
reduction to a corresponding one-dimensional problem. As before given a $k$-dimensional weight function $w$ and a $k$-dimensional vector $\vec{x}$ we denote by $w \cdot \vec{x}$ the one-dimensional weight function obtained by multiplying the weight vectors by $\vec{x}$. The reduction to one-dimensional objective requires the use of Gordan’s lemma.

**Lemma 63 (Gordan’s Lemma [52] (see also Lemma 2 in [77])).** For a matrix $A$, either $A \cdot \vec{x} = \vec{0}$ has a non-trivial non-negative solution for $\vec{x}$, or there exists a vector $\vec{y}$ such that $\vec{y} \cdot A^T$ is negative in every dimension.

The next lemma suggests that we can reduce the multidimensional problem to a corresponding one-dimensional problem.

**Lemma 64.** Given a WPS $A$ with a $k$-dimensional weight function $w$, and $(\gamma, q) \in \Gamma \times Q$, there exists a non-trivial non-negative solution for $PumpMat((\gamma, q)) \cdot \vec{x} = \vec{0}$ if and only if for every $\vec{x} \in \mathbb{R}^k$ there is a non-decreasing path from $(\gamma, q)$ to $(\gamma, q)$ that contains a pumpable pair $P = (p_1, p_2)$ such that $(w \cdot \vec{x})(P) \geq 0$ (i.e., the weight of the path for one-dimensional weight function $w \cdot \vec{x}$ is non-negative).

**Proof.** The proof is straightforward application of Gordan’s Lemma to the matrix $PumpMat((\gamma, q))$.

**Proposition 4.** There is a polynomial time algorithm that given WPS $A$ with $k$-dimensional weight function $w$, $(\gamma, q) \in \Gamma \times Q$, a vector $\vec{\lambda} \in Q^k$, and a rational number $r \in \mathbb{Q}$ decides if there exists a pumpable pair of paths $P$ in a non-decreasing cyclic path that begins at $(\gamma, q)$ in $A$, with $\frac{(w \cdot \vec{x})(P)}{|P|} > r$ and $|P| \leq \ell$, and if such pair exists, it returns $\frac{|P|}{w(P)}$.

Intuitively, the algorithm for Proposition 4 is based on the algorithm for solving WPSs with one-dimensional mean-payoff objectives. We postpone the technically detailed proof to Section 7.2.1. We first show how to use the result of the proposition and a result from linear programming to solve the problem. We first state the result for linear programming.

**Linear program with exponential constraints and polynomial-time separating oracle.** Consider a linear program over $n$ variables and exponentially many constraints in $n$. Given a polynomial time separating oracle that for every point in space returns in polynomial time whether the point is feasible, and if infeasible returns a violated constraint, the linear program can be solved in polynomial time using the ellipsoid method [53]. We use the result to show the following result.

**Proposition 5.** There exists a polynomial time algorithm that decides whether for a given state $q$ and a stack alphabet symbol $\gamma$ there exists a non-trivial non-negative solution for $PumpMat((\gamma, q)) \cdot \vec{x} = \vec{0}$.

**Proof.** Conceptually, given $q$ and $\gamma$, we compute a matrix $A$, such that each row in $A$ corresponds to the average weight vector of a row in $PumpMat((\gamma, q))$ (that is, the
weight of a pumpable pair divided by its length), and solves the following linear programming problem: For variables $r$ and $\vec{\lambda} = (\lambda_1, \ldots, \lambda_k)$, the objective function is to minimize $r$ subject to the constraints below:

$$\vec{\lambda} \cdot A^T \leq \vec{r} \quad \text{where} \quad \vec{r} = (r, r, \ldots, r)^T$$

(7.4)

$$\sum_{i=1}^{k} \lambda_i = 1$$

(7.5)

Once the minimal $r$ is computed, by Lemma 64, there exists a solution for $\text{PumpMat}((\gamma, q)) \cdot \vec{x} = 0$ if and only if $r \geq 0$.

The number of rows of $A$ in the worst case is exponential (to be precise at most $\ell \cdot (2 \cdot W \cdot \ell)^k$, since the length of the path is at most $\ell$, the sum of weights is between $-W \cdot \ell$ and $W \cdot \ell$ and there are $k$ dimensions). However, we do not enumerate the constraints of the linear programming problem explicitly but use the result of linear programs with polynomial time separating oracle. By Proposition 4 we have an algorithm that verifies the feasibility of a solution (that is, an assignment for $\vec{\lambda}$ and $r$) and if the solution is infeasible it returns a constraint that is not satisfied by the solution. Thus the result of Proposition 4 provides the desired polynomial-time separating oracle and we have the desired result.

Hence, we get the following theorem.

**Theorem 15.** Given a WPS $A$ with multidimensional weight function $w$, we can decide in polynomial time whether there exists a path $\pi$ such that $\text{LimAvg}(\pi) \geq \vec{0}$.

### 7.2.1 Technical detailed proof of Proposition 4

In this section we prove Proposition 4. Throughout this section, we assume WLOG that $\vec{\lambda}$ is a vector of integers and that $r = 0$. Intuitively the solution is very similar to solving WPS with one-dimensional objectives, with some technical and tedious modifications.

We will present the relevant details. Let $A$ be a WPS with $k$-dimensional weight function $w$, and $w \cdot \vec{\lambda}$ be the one-dimensional weight function. Let $d = ((|Q| \cdot |\Gamma|)^2 + 1$.

Before presenting the key lemma we recall the computation of the bounded height summary function $s_{i+1}$ from $s_i$ that will also introduce the relevant notions required for the lemma.

**Computation of $s_{i+1}$ from $s_i$ and $A$.** Let $G_A$ be the finite weighted graph that is formed by all the configurations of $A$ with stack height either one or two, that is, the vertices are of the form $(\alpha, q)$ where $q \in Q$ and $\alpha \in \{\bot \cdot \gamma, \bot \cdot \gamma_1 \cdot \gamma_2 | \gamma, \gamma_1, \gamma_2 \in \Gamma\}$. The edges (and their weights) are according to the transitions of $A$: formally, (i) (Skip edges): for vertices $(\bot \cdot \alpha, q)$ we have an edge to $(\bot \cdot \alpha, q')$ iff $e = (q, \text{Top}(\alpha), \text{skip}, q')$ is an edge in $A$ (and the weight of the edge in $G_A$ is $(w \cdot \vec{\lambda})(e)$) where $\alpha = \gamma$ or $\alpha = \gamma_1 \cdot \gamma_2$ for $\gamma, \gamma_1, \gamma_2 \in \Gamma$; (ii) (Push edges): for vertices $(\bot \cdot \gamma, q)$ we have an edge to $(\bot \cdot \gamma \cdot \gamma', q')$ iff $e = (q, \gamma, \text{push}(\gamma'), q')$ is an edge in $A$ (and the weight of the edge in $G_A$ is $(w \cdot \vec{\lambda})(e)$)
for \( \gamma, \gamma' \in \Gamma \); and (iii) (Pop edges): for vertices \((\bot \cdot \gamma \cdot \gamma', q)\) we have an edge to \((\bot \cdot \gamma \cdot q')\) iff \(e = (q, \gamma', \text{pop}, q')\) is an edge in \(A\) (and the weight of the edge in \(G_A\) is \((w \cdot \bar{\lambda})(e)\)) for \(\gamma, \gamma' \in \Gamma\). Intuitively, \(G_A\) allows skips, push pop pairs, and only one additional push. Note that \(G_A\) has at most \(2 \cdot |Q| \cdot |\Gamma|^2\) vertices, and can be constructed in polynomial time.

For every \(i \geq 1\), given the function \(s_i\), the graph \(G_A^i\) is constructed from \(G_A\) as follows: adding edges \(((\bot \gamma_1 \gamma_2, q_1), (\bot \gamma_1 \gamma_2, q_2))\) (if the edge does not exist already) and changing its weight to \(s_i(q_1, \gamma_2, q_2)\) for every \(\gamma_1, \gamma_2 \in \Gamma\) and \(q_1, q_2 \in Q\). The value of \(s_{i+1}(q_1, \gamma, q_2)\) is exactly the weight of the maximum weight path between \((\bot \gamma, q_1)\) and \((\bot \gamma, q_2)\) in \(G_A^i\) (with the following convention: \(-\infty < z < \omega, z + \omega = \omega\) and \(z + -\infty = \omega + -\infty = -\infty\) for every \(z \in \mathbb{Z}\)). If in \(G_A^i\) there is a path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) that contains a cycle with positive weight, then we set \(s_{i+1}(q_1, \gamma, q_2) = \omega\).

Hence, given \(s_i\) and \(A\), the construction of \(G_A^i\) is achieved in polynomial time, and the computation of \(s_{i+1}\) is achieved using the Bellman-Ford algorithm \([41]\) in polynomial time (the maximum weight path is the shortest weight if we define the edge length as the negative of the edge weight). Also note that the Bellman-Ford algorithm reports cycles with positive weight (that is, negative length) which is required to set \(\omega\) values of \(s_{i+1}\). It follows that we can compute \(s_{i+1}\) given \(s_i\) and \(A\) in polynomial time. In the computation of the summary function \(s_i\) we also store along with \(s_i(q_1, \gamma, q_2)\) the weight vector \(w(P)\) and the length \(|P|\) of a witness path \(P\) that is maximal weight (according to \(w \cdot \bar{\lambda}\)) shortest non-decreasing path from \((\gamma, q_1)\) to \((\gamma, q_2)\) with additional stack height at most \(i\). We denote by \(\text{VECT}(s_i(q_1, \gamma, q_2))\) the tuple \((w(P), |P|)\).

**Lemma 65.** Let \(q_1, q_2 \in Q\), \(\gamma \in \Gamma\) and \(d > (|Q| \cdot |\Gamma|)^2\), such that \(s_d(q_1, \gamma, q_2) > s_{d-1}(q_1, \gamma, q_2)\), and let \(\pi\) be the shortest non-decreasing path from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\) with weight \(s_{d+1}(q_1, \gamma, q_2)\) and additional stack height \(d\), then the following assertions hold:

1. The path \(\pi\) contains a pumpable pair of paths \(P = (p_1, p_2)\) with \((w \cdot \bar{\lambda})(P) > 0\) with length at most \(\ell\).

2. We can compute \(w(P)\), and \(\frac{w(P)}{|P|}\) in polynomial time.

**Proof.** The first item was proved in Section 7.1. For the second item, we consider the graphs \(G_A^d\) as defined above. Then for \(G_A^d\), we compute (based on the summary function \(s_d\)) the maximum weight non-decreasing path \(\rho\) from \((\bot \gamma, q_1)\) to \((\bot \gamma, q_2)\). In the path \(\rho\), we find a sub-path of the form \((\bot \gamma, z), (\bot \gamma \delta, q'), (\bot \gamma \delta, q''), (\bot \gamma, z')\), for which

- \(s_d(z, \gamma, z') > s_{d-1}(z, \gamma, z')\); and
- \(s_{d-1}(q', \delta, q'') > s_{d-2}(q', \delta, q'')\);

(note that by definition such sub-path must exist). We store the value of the maximum weight paths from \((\bot \gamma, q_1)\) to \((\bot \gamma, z)\), and from \((\bot \gamma, z')\) to \((\bot \gamma, q_2)\). We also store the push and pop transitions and the corresponding vector of the weight function \(w\),
and repeat the process, recursively, for the maximum weight non-decreasing path from (δ, q') to (δ, q'') with ASH(d − 1). We end up with a description of length O(d) of the form
\[
\rho^* = (\perp, 1, q_1^1) \rightarrow^{\text{push}_1} (\perp, 1, q_2^1) \rightarrow^{\text{push}_2} (\perp, 1, q_2^2) \rightarrow \ldots \rightarrow^{\text{push}_d} (\perp, 1, q_2^d) \rightarrow^{\text{pop}_d} (\perp, 1, 1, q_3^d) \rightarrow \ldots \rightarrow^{\text{pop}_2} (\perp, 1, 1, q_3^2) \rightarrow^{\text{push}_3} (\perp, 1, 1, q_4^3) \rightarrow \ldots \rightarrow^{\text{push}_3} (\perp, 1, 1, q_4^3)
\]
where \(q_1^1 = q_1\), \(q_4^1 = q_2\) and \(\gamma_1 = \gamma\). Intuitively, the path \(\rho^*\) is decomposed as the path \(\rho_1 \text{ push}_1 \rho_2 \text{ push}_2 \cdots \text{ push}_d \rho_{d+1} \text{ pop}_1 \rho_{d+2} \cdots \text{ pop}_d \rho_{2, d+1}\), where the \(\rho_1\) realizes the value \(s_d(q_1^1, \gamma_1, q_2^1)\), \(\rho_2\) realizes the value \(s_{d-1}(q_2^2, \gamma_2, q_2^2)\) and so on; and similarly \(\rho_{d+1}\) realizes the value \(s_0(q_3^1, \gamma_3, q_2^2)\), \(\rho_{d+2}\) realizes the value \(s_1(q_3^{d-1}, \gamma_{d-1}, q_4^{d-1})\), \(\rho_{d+3}\) realizes the value \(s_2(q_3^{d-2}, \gamma_{d-2}, q_4^{d-2})\) and so on; and finally, \(\rho_{2, d+1}\) realizes \(s_d(q_3^1, \gamma_1, q_4^1)\).

Since \(d > (|Q| \cdot |\Gamma|)^2\), there must exist \(1 \leq i < j \leq d\), and \(h_1, h_2, h_3, h_4 \in \{1, \ldots, 4\}\) such that \(q_{h_1}^i = q_{h_2}^i, q_{h_3}^i = q_{h_4}^i, \gamma_i = \gamma_j, \) and the weight of the path from \((\perp, 1 \ldots \gamma_j, q_{h_1}^i)\) to \((\perp, 1 \ldots \gamma_j, q_{h_2}^i)\) plus the weight of the path from \((\perp, 1 \ldots \gamma_j, q_{h_3}^i)\) to \((\perp, 1 \ldots \gamma_j, q_{h_4}^i)\) is positive. We sequentially iterate over all such tuples of \(i, j, h_1, h_2, h_3, h_4\) in polynomial time, and a witness path \(P\) can be obtained as of the form of \(\rho^*\). The computation of \(w(P)\) and \(\frac{w(P)}{|P|}\) is obtained from the vector of the summary function, and the \text{push} and \text{pop} transitions along with the vector of weights according to \(w\) of such transitions, i.e.,

\[
(w(P), |P|) = \left( \sum_{i=1}^{d} w(\text{push}_i) + w(\text{pop}_i), 2 \cdot d \right) + \sum_{i=1}^{2d+1} (w(\rho_i), |\rho_i|) + \sum_{i=1}^{d+1} \text{VECT}(s_{d+1-i}(q_1^i, \gamma_i, q_2^i)) + \sum_{i=d+2}^{2d+1} \text{VECT}(s_{d+1-i}(q_3^i, \gamma_i, q_4^i)).
\]

Hence it follows that we can compute \(w(P)\) and \(\frac{w(P)}{|P|}\) in polynomial time and the proof follows. 

Our goal now is the computation of the \(\omega\) values of the summary function. To achieve the computation of \(\omega\) values we will define another summary function \(s^*\) and a new WPS \(\mathcal{A}^*\) such that certain cycles in \(\mathcal{A}^*\) will characterize the \(\omega\) values of the summary function. We now define the summary function \(s^*\) and the pushdown system \(\mathcal{A}^*\). Let \(d = (|Q| \cdot |\Gamma|)^2\). The new summary function \(s^*\) is defined as follows: if the values of \(s_d\) and \(s_{d+1}\) are the same then it is assigned the value of \(s_d\), and otherwise
the value $\omega$. Formally,

$$s^*(q_1, \gamma, q_2) = \begin{cases} 
s_d(q_1, \gamma, q_2) & \text{if } s_d(q_1, \gamma, q_2) = s_{d+1}(q_1, \gamma, q_2) \\
\omega & \text{if } s_d(q_1, \gamma, q_2) < s_{d+1}(q_1, \gamma, q_2). \end{cases}$$

The new WPS $A^*$ is constructed from $A$ by adding the following set of $\omega$-edges:

$$\{(q_1, \gamma, q_2, \text{skip}) \mid s^*(q_1, \gamma, q_2) = \omega\}.$$

We will now present the required polynomial-time algorithm for Proposition 4, and we present the algorithm for the case with $r = 0$ (and this is without loss of generality). The algorithm is similar to solution of WPS with one-dimensional objective of Section 7.1. The final ingredient is the notion of summary graph.

**Summary graph and positive simple cycles.** Given a WPS $A = \langle Q, \Gamma, q_0 \in Q, E \subseteq (Q \times \Gamma) \times (Q \times \text{Com}(\Gamma)), w \cdot \vec{\lambda} : E \rightarrow \mathbb{Z} \rangle$ and the summary function $s$, we construct the summary graph $\text{Gr}(A) = (V, E)$ of $A$ with a weight function $\overline{w} : E \rightarrow \mathbb{Z} \cup \{\omega\}$ as follows: (i) $V = Q \times \Gamma$; and (ii) $E = E_{\text{skip}} \cup E_{\text{push}}$ where $E_{\text{skip}} = \{((q_1, \gamma), (q_2, \gamma)) \mid s(q_1, \gamma, q_2) > -\infty\}$, and $E_{\text{push}} = \{((q_1, \gamma_1), (q_2, \gamma_2)) \mid (q_1, \gamma_1, q_2, \text{push}(\gamma_2)) \in E\}$; and (iii) for all $e = ((q_1, \gamma), (q_2, \gamma)) \in E_{\text{skip}}$ we have $\overline{w}(e) = s(q_1, \gamma, q_2)$, and for all $e \in E_{\text{push}}$ we have $\overline{w}(e) = (w \cdot \vec{\lambda})(e)$ (i.e., according to weight function of $A$). A simple cycle $C$ in $\text{Gr}(A)$ is a positive simple cycle iff one of the following conditions hold: (i) either $C$ contains an $\omega$-edge (i.e., edge labeled $\omega$ by $\overline{w}$); or (ii) the sum of the weights of the edges of the cycles according to $\overline{w}$ is positive. The summary functions and the summary graph can be constructed in polynomial time. The first step of the algorithm is to build the summary graph and to check if there is a path from $(\gamma, q)$ to $(\gamma, q)$ with a positive weight. We consider the following cases of existence of such a positive weight path.

1. If there is no such path, then there does not exist pumpable pair of paths $P = (p_1, p_2)$ with positive weight (i.e., there exists no pumpable pair $P$ with $(w \cdot \vec{\lambda})(P) > 0$).

2. We now consider the case when such a positive weight path exists. If such a path exist, we consider the path with maximum weight that is shortest (i.e., among the ones with maximum weight we choose a path that is shortest). We have two distinct cases.

   (a) We first consider the case when the path do not go through an $\omega$ edge.

   Then the path does not have a pumpable pair for the following reason: if the pumpable pair is positive, then the weight is not the maximum, and if the pumpable pair is non-negative, removing it ensures we obtain a maximum weight path with shorter length. Hence the length of the path is at most $\ell$. Since we have stored the vector of the summary function (which stores the weights according to $w$ and length of the witness paths) we compute the
weight of this path according to $w$ (and not according to $w \cdot \lambda$), and return the average weight of this path.

(b) Otherwise, the path goes through an $\omega$ edge in the summary graph. If there is an $\omega$ edge due to a proper cycle with positive weight, then we can detect this cycle in the construction of the summary graph and compute its average weight according to $w$ (since we have the vector of the summary function that stores the weight according to $w$). Otherwise, by Lemma 51, it follows that there is a non-decreasing path from $(\gamma, q)$ to $(\gamma, q)$ that has a non-decreasing sub-path from $(\delta, q_1)$ to $(\delta, q_2)$ and $s_{d+1}(q_1, \delta, q_2) > s_d(q_1, \delta, q_2)$. We have already described a polynomial time algorithm for finding such $q_1, q_2$ and $\delta$. Once we find $q_1, q_2$ and $\delta$, by Lemma 65, we can compute $w(P)$ and $\frac{w(P)}{|P|}$ in polynomial time.

The proof of Proposition 4 follows.

### 7.3 Mean-Payoff Pushdown Games

In this section we consider pushdown games with mean-payoff objectives. We will show that the problem of deciding the existence of a strategy (or even a finite-memory strategy) to ensure mean-payoff objectives in pushdown games is undecidable. The undecidability results will be obtained by a reduction from the universality problem of weighted automata, which is known to be undecidable [4, 65]. We start with the definition of weighted pushdown games.

**Weighted pushdown games (WPGs).** A weighted pushdown game (WPG) $G = (\mathcal{A}, (Q_1, Q_2))$ consists of a WPS $\mathcal{A}$ and a partition $(Q_1, Q_2)$ of the state space $Q$ of $\mathcal{A}$ into player-1 states $Q_1$ and player-2 states $Q_2$. A WPG defines an infinite-state game graph $(V, E)$ with partition $(V_1, V_2)$ of the vertex set $V$, where $V$ is the set of configurations of $\mathcal{A}$, and $V_1 = \{(\alpha, q) \in V \mid q \in Q_1\}$, $V_2 = \{(\alpha, q) \in V \mid q \in Q_2\}$ and $E$ is obtained from the transitions of $\mathcal{A}$. The initial vertex is the configuration $(\bot, q_0)$.

**Plays and strategies.** A play on $G$ (or equivalently on the infinite-state game graph) is played in the following way: a pebble (or token) is placed on the initial vertex; and in every round, if the pebble is currently on a player-1 vertex (a vertex in $V_1$), then he chooses an edge to follow, and moves the pebble accordingly; and if the current vertex is a player-2 vertex, he does likewise. The process goes on forever and generates an infinite play (an infinite path $\pi$ in the infinite graph of the game). A strategy for player 1 is a recipe to extend plays; formally, a strategy for player 1 is a function $\tau : V^* \times V_1 \rightarrow V$ such that for all $w \in V^*$ and $v \in V_1$ we have $(v, \tau(w \cdot v)) \in E$. Equivalently, a strategy for player 1 given a history of configurations (i.e., the sequence of configurations of the finite prefix of a play) ending in a player-1 state, chooses the successor configuration according to the transitions of $\mathcal{A}$. A play $\pi = v_1 v_2 \ldots$ is consistent with a strategy $\tau$ if for every $v_i \in V_1$ we have $v_{i+1} = \tau(v_1 v_2 \ldots v_i)$, i.e., the play is possible according to the strategy $\tau$. The definition of player-2 strategies is analogous. Informally a strategy can be viewed as a transducer that takes as input the sequence of transitions, and
outputs the transitions to be taken. A strategy is called a finite-memory strategy if there is a finite-state transducer to implement the strategy. Formally, a finite path in a WPG $G$ with WPS $A$ starts in the initial configuration and is a finite sequence of transitions in $A$, and a finite-memory strategy is a finite-state transducer that has the set of transitions as both the input and the output alphabet. Thus given a finite sequence of transitions, the strategy as the transducer outputs the next transition to be played.

Winning strategies. We will consider mean-payoff objectives, as already defined in the previous section. A player-1 strategy $\tau$ is a winning strategy if for every play $\pi$ consistent with $\tau$ we have $\text{LimInfAvg}(\pi) \geq 0$ (resp. $\text{LimInfAvg}(\pi) > 0$, $\text{LimSupAvg}(\pi) \geq 0$, $\text{LimSupAvg}(\pi) > 0$). In other words, a winning strategy for player 1 ensures the mean-payoff objective against all strategies of player 2. We are interested in the question of existence of a winning strategy, and the existence of a finite-memory winning strategy for player 1 in WPGs with mean-payoff objectives. Our undecidability results for WPGs with mean-payoff objectives will be obtained by a reduction from the non-universality problem of weighted finite automata. We define the problem below.

**Weighted finite automata (WFA).** A weighted finite automaton (WFA) is a tuple $A = (\Sigma, Q, q_0, \Delta, w : \Delta \to \mathbb{Z})$, where $\Sigma$ is a finite input alphabet, $Q$ is a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $w : \Delta \to \mathbb{Z}$ is a weight function and $q_0 \in Q$ is the initial state. For a word $\rho = \sigma_1 \sigma_2 \ldots \sigma_n$, a run of $A$ on $\rho$ is a sequence $r = r_0 r_1 \ldots r_n \in Q^*$, where $r_0 = q_0$, and for all $1 \leq i \leq n$ we have $d_i = (r_{i-1}, \sigma_i, r_i) \in \Delta$.

The weight of the run $r$ is $w(r) = \sum_{i=1}^n w(d_i)$. Since the automaton is non-deterministic there maybe several runs for a word, and the weight of a finite word $\rho \in \Sigma^*$ over $A$ is the minimal weight over all runs on $\rho$, i.e., $L_A(\rho) = \min \{ w(r) \mid r \text{ is a run of } A \text{ on } \rho \}$. The non-universality problem asks, given $\nu \in \mathbb{Z}$, whether there exists a word $\rho \in \Sigma^*$ for which $L_A(\rho) \geq \nu$?; equivalently, is it not the case that for every $\rho \in \Sigma^*$ we have $L_A(\rho) \leq \nu - 1$?

**Theorem 16 ([4]).** The non-universality problem is undecidable for WFA with weight functions $w : \Delta \to \{-1, 0, 1\}$.

Informally, given a WFA $A$ we will construct a WPG in such way that in the first rounds player-1 fills the stack with letters that construct a word $\rho$ of $A$, and then player-2 simulates the WFA’s minimal run on $\rho$ and then the game returns to the initial state. If for all $\rho \in \Sigma^*$ we have $L_A(\rho) \leq 0$, then the mean-payoff of the play will be at most 0, otherwise, there exists a word $\rho \in \Sigma^*$ such that $L_A(\rho) > 0$, and then by playing according to $\rho$, player-1 can ensure a positive mean-payoff.

**Reduction: WFA to WPGs.** We first prove that WPGs are undecidable for $\text{LimInfAvg}(\pi) > 0$ and $\text{LimSupAvg}(\pi) > 0$ objectives. This proof will immediately show the undecidability also for $\text{LimInfAvg}(\pi) \geq 0$ and $\text{LimSupAvg}(\pi) \geq 0$ objectives, as $\text{LimInfAvg}(\pi) \geq 0$ (resp. $\text{LimSupAvg}(\pi) \geq 0$) is dual to the objective of player 2 when the objective of player 1 is $\text{LimSupAvg}(\pi) > 0$ (resp. $\text{LimInfAvg}(\pi) > 0$).
Reduction. The reduction from the non-universality problem of a weighted automaton is as follows. Given a WFA $A = (\Sigma, Q, q_0, \Delta, w : \Delta \rightarrow \{-1, 0, 1\})$ we construct a WPG $G$ with the aid of five gadgets, and we describe the gadgets below. WLOG we assume that there is a special symbol $\$ that does not belong to $\Sigma$.

1. **Gadget 1.** The first gadget contains only one state, namely $q_\$$, which is a player-1 state. The state has two possible transitions. In the first transition it pushes $\$ into the stack and remains in the same state. In the second transition it pushes $\$ and goes to the second gadget. All the weights in this gadget are $-10$.

2. **Gadget 2.** The second gadget also contains one state, namely $q_{\Sigma}$, which is also a player-1 state. For every $\sigma \in \Sigma$ the state has a transition that pushes $\sigma$ into the stack and remains in the same state. In addition there is one more transition, which leads to the third gadget keeping the stack unchanged with weight 0. All the weights in this gadget (other than the skip transition) are $-1$. Informally, in this gadget player 1 needs to construct a word $\rho$ such that the reverse of $\rho$ has value at least 1 in $A$. For a word $\rho$, let $\text{rev}(\rho)$ denote the reverse of the word.
   - In this gadget player 1 should construct a word $\rho \in \Sigma^*$ for which $L_A(\text{rev}(\rho)) \geq 1$.
   - The WPG $G$ will be constructed in such way that player 1 must play in a way so that the number of $\$ in the stack will be greater than the number of letters from $\Sigma$ to ensure the mean-payoff objectives.

3. **Gadget 3.** The third gadget is the choice gadget with only one player-2 state $q_{ch}$, which either leads to the fourth gadget or the fifth gadget. The weights of the transitions are 0 and the stack is not changed. Informally, player-2 should go to the fifth gadget if the word that player 1 pushed into the stack has non-positive weight, and should go to the fourth gadget if the number of $\$ symbols in the stack is less than the number of symbols from $\Sigma$.

4. **Gadget 4.** The fourth gadget consists of only one player-2 state $q_{<\$}$ (to denote that there is not enough $\$ symbols). It has a transition $\text{pop}(\sigma)$ with 0 weight, for all $\sigma \in \Sigma$; and a transition $\text{pop}(\$)$ with $+11$ weight. If the stack is empty, then there is a transition to the initial state.
   A pictorial description of the first four gadgets is shown in Figure 7.5, where $\bigcirc$ denotes player-1 states, $\square$ denotes player-2 states, and edges are labeled by stack top; followed by the stack command; and then the weight; and if the stack top is irrelevant (i.e., the transition is valid for all stack tops), then it is denoted as $\_$.
   We now describe the fifth gadget.

5. **Gadget 5.** The fifth gadget is the simulate run gadget. The states in this gadgets are essentially the set $Q$ of states of the automaton $A$; and all the states are
player-2 states. The transitions and edge weights are as follows: (i) for every \((q, \sigma, q') \in \Delta\) we have a transition \((q, \sigma, q', \text{pop}(\sigma))\), with weight \(w_A(q, \sigma, q') + 1\) (1 plus the weight in \(A\)); and (ii) in addition there exists a transition \((q, $, q, \text{pop}($))\) with weight +10 and a transition \((q, \bot, q, \text{skip})\) to the initial state for empty stack with weight 0.

**Comment on naive solution.** Our reduction consists of five gadgets and we comment on a simpler reduction. Consider a naive solution with only two gadgets (Gadget 2 and Gadget 5): in one player 1 fills the stack and in the other player 2 simulates the run. This reduction only works for \(\text{LimInfAvg}\) objectives with strict inequality. However, for \(\text{LimInfAvg}(\pi) \geq 0\) objective, it is possible that the WFA assigns \(-1\) value for all finite words, but player 1 can achieve \(\text{LimInfAvg}(\pi) = 0\) by selecting words with increasing lengths. For the \(\text{LimSupAvg}(\pi) > 0\) objective it is possible that player 1 will select a finite word \(\rho\) such that there is a run of the WFA \(A\) that assigns a zero weight to \(\rho\), but the run has a prefix with positive weight. Hence, for infinitely many positions of the path of the two-player game the total weight is positive and hence the path satisfies \(\text{LimSupAvg}(\pi) > 0\) objective. To handle the above issues, we introduce Gadgets 1, 3, and 4. Intuitively, these gadgets ensure that before player 1 fills the stack with a word \(\rho\), he will make at least \(|\rho| + 1\) push $ transitions (with negative weights). Consequently, when player 2 simulates a run of the WFA the total weight of the play remains non-positive, at least until the pop $ transitions are done, and the weight becomes positive only if \(L_A(\rho) > 0\).
Correctness of reduction. We will now prove the correctness of the reduction by showing that there is a winning strategy (also a finite-memory winning strategy) in the WPG $G$ for mean-payoff objectives with strict inequality iff there is a finite word $\rho \in \Sigma^*$ such that $L_A(\rho) \geq 1$. Let $\pi$ be a play on the above WPG $G$. The $i$-th iteration of the play are the positions between the $i$-th visit and the $(i+1)$-th visit to the initial state.

**Lemma 66.** If there is a word $\rho \in \Sigma^*$ such that $L_A(\rho) \geq 1$, then there exists a finite-memory strategy $\tau_1^*$ for player 1 to ensure that for all plays $\pi$ consistent with $\tau_1^*$ we have $\operatorname{LimSupAvg}(\pi) > 0$ and $\operatorname{LimInfAvg}(\pi) > 0$.

**Proof.** The finite-memory strategy $\tau_1^*$ for player 1 is to play in every iteration $\$^{n+1}$ in $q_\$ \ast$ followed by $\ast$ in $q_\Sigma$, where $n = |\rho|$ is the length of the word $\rho$. In every iteration, the sum of the weights is at least 1 as $L_A(\rho) \geq 1$, and the length of a play in every iteration is at most $4 \cdot n + 2$. It follows that for all plays $\pi$ consistent with $\tau_1^*$ we have both $\operatorname{LimSupAvg}(\pi) > 0$ and $\operatorname{LimInfAvg}(\pi) > 0$.

**Lemma 67.** If for all words $\rho \in \Sigma^*$ we have $L_A(\rho) \leq 0$, then there exists a counter strategy $\tau_2^*$ for player 2 to ensure that for all strategies $\tau_1$ of player 1, for the play given $\tau_2^*$ and $\tau_1$: for all iterations $i$, for every position between the $i$-th iteration and the $(i+1)$-th iteration, the sum of the weights from the beginning of the iteration to the current position of the iteration is at most 0.

**Proof.** The counter strategy $\tau_2^*$ is as follows: consider an iteration $i$, and let the strategy of player 1 in this iteration produce the sequence $\$^{n}$$, for $\rho \in \Sigma^*$. Note that if the state $q_\text{ch}$ is never reached, then all the weights are negative (in $q_\$ \ast$ and $q_\Sigma$ all weights are negative). The strategy $\tau_2^*$ is described considering the following two cases.

1. If $n \leq |\rho|$, then the strategy $\tau_2^*$ chooses the state $q_{\$less}$ at the state $q_\text{ch}$ (since there are not enough $\$$ in the stack). For any position of the play till state $q_\text{ch}$ is reached, the sum of the weights is negative. Once $q_{\$less}$ is reached, for any position, the payoff is at most $-10 \cdot n - |\rho| + 11 \cdot n = -|\rho| + n \leq 0$, as $n \leq |\rho|$.

2. Otherwise, we have $n > |\rho|$ and $L_A(\ast)$ $\leq 0$. There exists a run $r$ on $\ast$ such that for every prefix $\beta$ of $\ast$ the sum of the weights is at most $2 \cdot |\beta| \leq 2 \cdot |\rho| < 2 \cdot n$ (since the absolute value of the weights of $A$ are bounded by 1 and therefore the weights in Gadget 5 are bounded by 2) and in the end of the run, the sum of the weights is at most 0. The counter strategy $\tau_2^*$ follows the run $r$. Hence the sum of the weights for any prefix $\beta$ is at most $-10 \cdot n + 2 \cdot |\beta| \leq -10 \cdot n + 2 \cdot n < 0$, until the letter $\$$ is the top symbol of the stack. Once $\$$ is the top symbol, the sum of the weights is at most $-10 \cdot n$, since the sum of the weights of the run is at most 0. Since with each pop of $\$$ the weight is 10, and there are $n$ pops, it follows that in every position of the iteration the sum of the weights is at most 0. Finally, once the iteration is completed the sum of the weights is also at most 0.
The desired result follows.

Lemma 68. Given WFA $A$ and the WPG $G$ constructed by the reduction, the following assertions hold:

1. If there is a word $\rho \in \Sigma^*$ such that $L_A(\rho) \geq 1$, then there is a finite-memory winning strategy $\tau_1^*$ for player 1 for the objectives $\text{LimSupAvg}(\pi) > 0$ and $\text{LimInfAvg}(\pi) > 0$.

2. If for all words $\rho \in \Sigma^*$ we have $L_A(\rho) \leq 0$, then there is no winning strategy for player 1 for the objectives $\text{LimSupAvg}(\pi) > 0$ and $\text{LimInfAvg}(\pi) > 0$.

3. There exists a winning strategy (resp. a finite-memory winning strategy) for player 1 for the objectives $\text{LimSupAvg}(\pi) > 0$ and $\text{LimInfAvg}(\pi) > 0$ iff there is a word $\rho \in \Sigma^*$ such that $L_A(\rho) \geq 1$.

Proof. Note that the third item is a consequence of the first two items. The first item follows from Lemma 66. We now use Lemma 67 to prove the second item. Given the condition of the second item, let us consider the strategy $\tau_2^*$ for player 2 as described in Lemma 67. Let $\pi$ be a play consistent with $\tau_2^*$. We consider two cases to complete the proof.

- If $\pi$ does not have an infinite number of iterations, then from some point on only states $q_8$ or $q_\Sigma$ are visited, and they both have only negative weights. Hence all the weights that occur in $\pi$ from some point on are non-positive and hence $\text{LimInfAvg}(\pi) \leq \text{LimSupAvg}(\pi) \leq 0$.

- Otherwise, $\pi$ has an infinite number of iterations. Given $\tau_2^*$ it follows from Lemma 67 that for all iterations, in every position of an iteration, the sum of the weights from the beginning of the iteration to the current position is non-positive. Hence $\text{LimInfAvg}(\pi) \leq \text{LimSupAvg}(\pi) \leq 0$.

The desired result follows.

Undecidability for related decision problems. It follows from Lemma 68 that the existence of winning strategies (resp. finite-memory winning strategies) for mean-payoff objectives with strict inequality is undecidable for WPGs. For general strategies the result also follows for non-strict inequality by duality. We now show the undecidability for finite-memory strategies for the non-strict inequality as well as undecidability for stack boundedness. This is done by showing a reduction from the non-universality problem for WFA with threshold $\nu = 0$. The reduction is identical to the original reduction presented in this section. If there exists a word $\rho \in \Sigma^*$ such that $L_A(\rho) \geq 0$, then playing $\$|\rho|+1\text{rev}(\rho)$ in every iteration is a finite-memory winning strategy for
player-1 (also for the stack boundedness condition). Otherwise, for every \( \rho \in \Sigma^* \) we have \( L_A(\rho) \leq -1 \). In this case, against every player-1 finite-memory strategy, with memory size \( M \), player-2 has a strategy that ensures that the mean-payoff is at most \(-\frac{1}{2M}\). Similarly, against every player-1 strategy that ensures stack height at most \( M \), player-2 has a strategy that ensures that the mean-payoff at most \(-\frac{1}{2M}\) (since every iteration is of length at most \( 2M \) and the total weight of every iteration is at most \(-1\)).

**Comment on threshold and unique initial state.** We note that the non-universality problem for WFA remains undecidable for the special threshold 0 and a unique initial state. This is because given a WFA \( A \) with a set of initial states and threshold \( \nu \), we construct a WFA \( A' \) with a newly added unique initial state that has all the transitions of the set of initial states of \( A \) and the weights of the newly added transitions (of which exactly one is executed exactly once) are the original weights added with \(-\nu\). The non-universality problem of \( A \) with threshold \( \nu \) coincide with the non-universality of \( A' \) with threshold 0.

**Theorem 17.** Given a WPG \( G \), the following questions are undecidable: (1) Whether there exists a winning strategy (resp. finite-memory winning strategy) to ensure \( \Phi \bowtie 0 \), where \( \Phi \in \{\text{LimSupAvg}, \text{LimInfAvg}\} \) and \( \bowtie \in \{\geq, >\} \); and (2) whether there exists a winning strategy (resp. finite-memory winning strategy) to ensure \( \Phi \bowtie 0 \) along with stack boundedness, where \( \Phi \in \{\text{LimSupAvg}, \text{LimInfAvg}\} \) and \( \bowtie \in \{\geq, >\} \).

**Distinguishing facts.** We now show some interesting facts about WPGs with mean-payoff objectives that distinguish them from finite game graphs with mean-payoff objectives.

1. **Fact 1.** It follows from the proof of Theorem 12 (Example 6 for the fact that ultimately periodic paths are not sufficient) that in general, positional (or memoryless) strategies are not sufficient, and infinite-memory strategies are required in general (in contrast, in finite game graphs, memoryless winning strategies are guaranteed to exist).

2. **Fact 2.** The objectives \( \text{LimSupAvg} \) and \( \text{LimInfAvg} \) do not coincide in general for WPGs. We show this in Example 7.

3. **Fact 3.** We also note that pushdown mean-payoff games are very different as compared to parity games. For finite-state games both parity and mean-payoff objectives have the same complexity (both lie in \( \text{NP} \cap \text{coNP} \) and also in \( \text{UP} \cap \text{coUP} \) [58]), in contrast for pushdown games the mean-payoff problem is undecidable, whereas the parity problem is \( \text{EXPTIME-complete} \) [102]. Moreover, for countably infinite games with finitely many priorities, for parity objectives memoryless winning strategies exist [91], whereas as we show (Fact 1) for mean-payoff pushdown games infinite-memory strategies are required.
Example 7. We show that there exists a WPG such that player-1 can ensure that 
\( \limsup \text{Avg}(\pi) \geq 2 \) and player-2 can ensure that \( \liminf \text{Avg}(\pi) \leq -2 \). The WPG is 
described as follows: Let \( Q_1 = \{ q^1_1, q^2_2 \} \) and \( Q_2 = \{ q^1_1, q^2_2 \} \), and let \( E \), the set of 
transitions, be as follows

- \( (q^1_1, \bot, q^1_1, \text{push}(\gamma)) \) with weight \(-2\);
- \( (q^1_1, \gamma, q^1_1, \text{push}(\gamma)) \) with weight \(-2\);
- \( (q^1_1, \gamma, q^2_2, \text{skip}) \) with weight \(0\);
- \( (q^2_2, \gamma, q^2_2, \text{pop}) \) with weight \(+4\);
- \( (q^2_2, \bot, q^1_1, \text{skip}) \) with weight \(0\);
- \( (q^1_1, \bot, q^1_1, \text{push}(\gamma)) \) with weight \(+2\);
- \( (q^1_1, \gamma, q^1_1, \text{push}(\gamma)) \) with weight \(+2\);
- \( (q^1_1, \gamma, q^2_2, \text{skip}) \) with weight \(0\);
- \( (q^2_2, \gamma, q^2_2, \text{pop}) \) with weight \(-4\);
- \( (q^2_2, \bot, q^1_1, \text{skip}) \) with weight \(0\).

The WPG is shown in Figure 7.6. It is straightforward to verify that player-1 can 
ensure \( \limsup \text{Avg}(\pi) \geq 2 \), and player-2 can ensure \( \liminf \text{Avg}(\pi) \leq -2 \).
Chapter 8

Pushdown Mean-Payoff Games with Modular Strategies

8.1 Recursive Games and Modular Strategies

In this section we will consider modular strategies in pushdown games, and modular strategies are more intuitive in the equivalent model of recursive game graphs. We first present an overview of the solution, and then the definition of recursive game graphs from [9].

Overview of the solution. We first show that player 1 has a modular winning strategy that is cycle free, namely, it is does not depend on the simple cycles that occur in the history of the play. We then show that a cycle-free strategy can be simulated by a memoryless strategy and by the results of Section 7.1 we get that mean-payoff modular games are in NP (as we have a polynomial-time verifier for a player-1 winning strategy, namely, a verifier for a memoryless winning strategy). The NP-hardness is obtained via a reduction from the 3-SAT problem.

Weighted recursive game graphs (WRGs). A recursive game graph $A$ consists of a tuple $(A_1, \ldots, A_n)$ of game modules, where each game module $A_i = (N_i, B_i, V_1^i, V_2^i, \text{En}_i, \text{Ex}_i, \delta_i)$ consists of the following components:

- A finite nonempty set of nodes $N_i$.
- A nonempty set of entry nodes $\text{En}_i \subseteq N_i$ and a nonempty set of exit nodes $\text{Ex}_i \subseteq N_i$.
- A set of boxes $B_i$.
- Two disjoint sets $V_1^i$ and $V_2^i$ that partition the set of nodes and boxes into two sets, i.e., $V_1^i \cup V_2^i = N_i \cup B_i$ and $V_1^i \cap V_2^i = \emptyset$. The set $V_1^i$ (resp. $V_2^i$) denotes the places where it is the turn of player 1 (resp. player 2) to play (i.e., choose transitions). We denote the union of $V_1^i$ and $V_2^i$ by $V_i$.
- A labeling $Y_i : B_i \rightarrow \{1, \ldots, n\}$ that assigns to every box an index of the game modules $A_1 \ldots A_n$.

134
Let \( \text{Calls}_i = \{(b, e) \mid b \in B_i, e \in \text{En}_j, j = Y_i(b)\} \) denote the set of calls of module \( A_i \), and let \( \text{Retns}_i = \{(b, x) \mid b \in B_i, x \in \text{Ex}_j, j = Y_i(b)\} \) denote the set of returns in \( A_i \). Then, \( \delta_i \subseteq (N_i \cup \text{Retns}_i) \times (N_i \cup \text{Calls}_i) \) is the transition relation for module \( A_i \).

A weighted recursive game graph (for short WRG) is a recursive game graph, equipped with a weight function \( w \) on the transitions. We also refer the readers to [9] for detailed description and illustration with figures of recursive game graphs. WLOG we shall assume that the boxes and nodes of all modules are disjoint. Let \( B = \bigcup_i B_i \) denote the set of all boxes, \( N = \bigcup_i N_i \) denote the set of all nodes, \( \text{En} = \bigcup_i \text{En}_i \) denote the set of all entry nodes, \( \text{Ex} = \bigcup_i \text{Ex}_i \) denote the set of all exit nodes, \( V^1 = \bigcup_i V^1_i \) (resp. \( V^2 = \bigcup_i V^2_i \)) denote the set of all places under player 1’s control (resp. player 2’s control), and \( V = V^1 \cup V^2 \) denote the set of all vertices. We will also consider the special case of one-player WRGs, where either \( V^2 \) is empty (player-1 WRGs) or \( V^1 \) is empty (player-2 WRGs).

Configurations, paths, and local history. A configuration \( c \) consists of a sequence \( (b_1, \ldots, b_r, u) \), where \( b_1, \ldots, b_r \in B \) and \( u \in N \). Intuitively, \( b_1, \ldots, b_r \) denote the current stack (of modules), and \( u \) is the current node. A sequence of configurations is valid if it does not violate the transition relation. The configuration stack height of \( c \) is \( r \). Let us denote by \( \mathcal{C} \) the set of all configurations, and let \( \mathcal{C}_1 \) (resp. \( \mathcal{C}_2 \)) denote the set of all configurations under player 1’s control (resp. player 2’s control). A path \( \pi = (c_1, c_2, c_3, \ldots) \) is a valid sequence of configurations. Let \( \rho = \langle c_1, c_2, \ldots, c_k \rangle \) be a valid finite sequence of configurations, such that \( c_i = (b_{i_1}, \ldots, b_{i_m}, u_i) \), and the stack height of \( c_i \) is \( d_i \). The stack height of \( \pi \), denoted by \( \text{SH}(\pi) \), is \( \max\{d_1, \ldots, d_k\} \) and \( \text{ASH}(\pi) = \text{SH}(\pi) - \max\{d_1, d_k\} \). Let \( c_i \) be the first configuration with stack height \( d_i = d_k \), such that for every \( i \leq j \leq k \), if \( c_j \) has stack height \( d_i \), then \( u_j \notin \text{Ex} \) (\( u_j \) is not an exit node). The local history of \( \rho \), denoted by \( \text{LocalHistory}(\rho) \), is the sequence \( (u_{j_1}, \ldots, u_{j_m}) \) such that \( c_{j_i} = c_i, c_{j_m} = c_k, j_1 < j_2 < \cdots < j_m \), and the stack height of \( c_{j_1}, \ldots, c_{j_m} \) is exactly \( d_i \). Intuitively, the local history is the sequence of nodes in a module. Note that by definition, for every \( \rho \in \mathcal{C}^* \), there exists \( i \in \{1, \ldots, n\} \) such that all the nodes that occur in \( \text{LocalHistory}(\rho) \) belongs to \( V_i \). We say that \( \text{LocalHistory}(\rho) \in A_i \) if all the nodes in \( \text{LocalHistory}(\rho) \) belongs to \( V_i \).

Global game graph and isomorphism to pushdown game graphs. The global game graph corresponding to a WRG \( \mathcal{A} = (A_1, \ldots, A_n) \) is the graph of all valid configurations, with an edge \( (c_1, c_2) \) between configurations \( c_1 \) and \( c_2 \) if there exists a transition from \( c_1 \) to \( c_2 \). It follows from the results of [9] that every recursive game graph has an isomorphic pushdown game graph that is computable in polynomial time (namely, there is an efficient algorithm to transform the finite representation of a recursive state machine to an equivalent finite representation of a pushdown graph and vice-versa). We note that the simulation requires some extra transitions that may influence the value of mean-payoff in the original and the simulated runs. But since we only ask whether the mean-payoff value is positive (or non-negative), this does not influence our results,
since we assign zero weights for the auxiliary transitions.

**Plays, strategies, and modular strategies.** A play begins at the entry node of module $A_0$ and it is played in the usual sense over the global game graph (which is possibly an infinite graph). A (finite) play is a (finite) valid sequence of configurations $(c_1, c_2, c_3, \ldots)$ (i.e., a path in the global game graph). A finite path $\pi$ is a sub-play if there exist a finite path $\pi_0$ such that $\pi_0 \cdot \pi$ is a prefix of a valid play. A strategy for player 1 is a function $\tau : C^* \times C_1 \to C$ respecting the edge relationship of the global game graph, i.e., for all $w \in C^*$ and $c_1 \in C_1$ we have that $(c_1, \tau(w \cdot c_1))$ is an edge in the global game graph. A modular strategy $\tau$ for player 1 is a set of functions $\{\tau_i\}_{i=1}^n$, one for each module, where for every $i$, we have $\tau_i : (N_i \cup \text{Retns})^* \to \delta_i$. The function $\tau$ is defined as follows: For every play prefix $\rho$ we have $\tau(\rho) = \tau_i(\text{LocalHistory}(\rho))$, where $\text{LocalHistory}(\rho) \in A_i$. The function $\tau_i$ is the local strategy of module $A_i$. Intuitively, a modular strategy only depends on the local history, and not on the context of invocation of the module. A modular strategy $\tau = \{\tau_i\}_{i=1}^n$ is a finite-memory modular strategy if $\tau_i$ is a finite-memory strategy for every $i \in \{1, \ldots, n\}$. A memoryless modular strategy is defined in similar way, where every component local strategy is memoryless.

**Mean-payoff objectives and winning modular strategies.** The weight of a finite path $\pi$, denoted by $w(\pi)$ is the sum of all the weights along the path. For an infinite path $\pi$ (as in the previous sections) we denote $\liminfavg(\pi) = \liminf_{n \to \infty} \frac{w(\pi[1, n])}{n}$ (resp. $\limsupavg(\pi) = \limsup_{n \to \infty} \frac{w(\pi[1, n])}{n}$), where $\pi[1, n]$ is the initial prefix of length $n$. The modular winning strategy problem asks if player 1 has a modular strategy $\tau$ such that for every play $\rho$ consistent with $\tau$ we have $\liminfavg(\rho) \geq 0$ (note that the counter strategy of player 2 is a general strategy), and similarly for other mean-payoff objectives.

**Basic properties.** We now present some basic properties of recursive game graphs.

**Non-decreasing cycles and proper cycles.** A non-decreasing cycle in a recursive game graph $A = \langle A_1, \ldots, A_n \rangle$ is a path segment from a module $A_i$ and vertex $v_i \in A_i$ to the same module and the same vertex (possibly at different stack level), such that the first occurrence of module $A_i$ in the path segment does not return (i.e., does not reach an exit node) during the path segment. A non-decreasing cycle $C$ is a proper cycle if the stack heights at the beginning and the end of the path segment are the same.

**Lemma 69.** Consider a one-player WRG $A = \langle A_1, \ldots, A_n \rangle$ (i.e., consists of only one-player). The following assertions hold:

- The WRG $A$ has a path $\pi$ with $\liminfavg(\pi) > 0$ (resp. $\limsupavg(\pi) > 0$) iff there exists a positive non-decreasing cycle.

- The WRG $A$ has a path $\pi$ with $\liminfavg(\pi) < 0$ (resp. $\limsupavg(\pi) < 0$) iff there exists a negative non-decreasing cycle.

**Proof.** The first item follows from (i) the isomorphism of one-player WRGs and weighted pushdown systems (WPSs), (ii) the correspondence of positive non-decreasing
cycles and good cycles for WPSs, and (iii) the results established in Section 7.1 showing the equivalence of the existence of a path $\pi$ with $\text{LimInfAvg}(\pi) > 0$ (resp. $\text{LimSupAvg}(\pi) > 0$) and the existence of good cycles in a WPS. The second item follows from the duality of $\text{LimInfAvg}(\pi) > 0$ and $\text{LimSupAvg}(\pi) < 0$.

**Lemma 70.** Given a WRG $A$, a modular strategy $\tau$, every (finite or infinite) path in the one-player WRG $A^{\tau}$ is a (finite or infinite) play in $A$ consistent with $\tau$, and vice versa.

**8.1.1 Decidability of the modular winning strategy problem**

In this section we will establish the decidability of the existence of a modular winning strategy problem. In the following section we will establish the NP upper bound, and finally show NP-hardness. We start with the objective $\text{LimInfAvg} \geq 0$, and then show the result for the objective $\text{LimSupAvg} \geq 0$. The results for mean-payoff objectives with strict inequality will also easily follow from our results.

**Remark 2.** We note that the reduction we presented in Section 7.3 from the non-universality problem of WFA to two-player mean-payoff pushdown games does not hold when we restrict player 1 to modular strategies. Indeed, a modular strategy cannot fill the stack with an arbitrary long stack alphabet string without eventually visiting the same module twice (and then it must be the case that the operations are repeated forever). Hence, player 1 cannot fill the stack with an arbitrary word to witness the non-universality of the WFA.

**Objective $\text{LimInfAvg} \geq 0$.** For the decidability result, we will show the existence of cycle independent modular winning strategies, and the result will also be useful to establish the complexity results. We start with the notion of a cycle free path in a graph.

**Cycle free path.** Let $G = (V, E)$ be a simple (no parallel edges) directed graph. We define the operator $\text{CycleFree} : V^* \rightarrow V^*$ in the following way: let $\pi = \langle v_1, v_2, \ldots, v_n \rangle$ be a finite path in $G$.  

**WRG given finite-memory strategies.** Given a WRG $A$, let $\tau = \{\tau_i\}_{i=1}^n$ be a finite-memory modular strategy. Let $M_i$ be the set of memory states of strategy $\tau_i$, i.e., $\tau_i$ is described as a deterministic transducer with state space $M_i$. The one-player WRG (player-2 WRG) given $\tau$ is the tuple $A^{\tau} = \langle A^{\tau}_1 = A_1 \times M_1, \ldots, A^{\tau}_n = A_n \times M_n \rangle$, where each $A^{\tau}_i = A_i \times M_i$ is obtained as the synchronous product of $A_i$ and the deterministic transducer describing the local strategy $\tau_i$. Formally, in the product, if the second component is a state $x_i \in M_i$, then the transition for the second component is as defined by the transition function of the deterministic transducer over $x_i$, and in the first component transition we only retain the transition prescribed by $x_i$. The weights of the transitions are specified according to the weight function of $A$. Note that if $\tau$ is a memoryless modular strategy, then $A^{\tau}$ is a sub-game graph of $A$. 

*CHAPTER 8. PUSHDOWN MEAN-PAYOFF GAMES WITH MODULAR STRATEGIES* 137
• CycleFree(\(\pi\)) = \(\pi\) if \(\pi\) is a simple path (i.e., with no cycles).

• Otherwise we define CycleFree inductively as follows. Let CycleFree(\(v_1 \ldots v_{n-1}\)) = \(u_1 u_2 \ldots u_m\). Let \(i\) be the first index such that \(v_n = u_i\). If such an index does not exist, then CycleFree(\(\pi\)) = \(u_1 u_2 \ldots u_m v_n\). Otherwise CycleFree(\(\pi\)) = \(u_1 u_2 \ldots u_i\).

Intuitively, the CycleFree operator takes a finite path and returns a simple path by removing the simple cycles according to the order of appearance.

**Cycle independent modular strategy.** Given a recursive game graph, a local strategy \(\tau\) is a cycle independent modular winning strategy. To establish the result we introduce the notion of manipulated paths, using rewind, fast forward, and simulation operations.

**Manipulated paths, rewind, fast forward, and simulation operations.** Let \(\tau = \{\tau_i\}_{i=1}^n\) be a modular winning strategy for the objective \(\text{LimInfAvg} \geq 0\), and let \(\epsilon > 0\) be an arbitrary constant. Let \(\pi_m = \pi_{m-1} \cdot n_i\) be a play prefix at position \(m\), that ends at node \(n_i \in A_i\). The manipulated play prefix of \(\pi_m\) according to \(\tau\) and \(\epsilon\), denoted by \(\text{Man}_\epsilon^\tau(\pi_m)\), is defined inductively as follows: Let \(\text{Man}_\epsilon^\tau(\pi_{m-1})\) be the manipulated play prefix at position \(m-1\). Then \(\text{Man}_\epsilon^\tau(\pi_m)\) is obtained from \(\text{Man}_\epsilon^\tau(\pi_{m-1})\) and \(n_i\) by one of the following operations.

1. **Rewind operation:** The condition for the rewind operation is that CycleFree(\(\text{Man}_\epsilon^\tau(\pi_{m-1})\)) \(\cdot n_i\) closes a proper cycle in the top module \(A_i\). If the rewind condition holds, then \(\text{Man}_\epsilon^\tau(\pi_m)\) is formed from \(\text{Man}_\epsilon^\tau(\pi_{m-1}) \cdot n_i\) by removing the proper cycle suffix from \(\text{Man}_\epsilon^\tau(\pi_{m-1}) \cdot n_i\). Intuitively the rewind operation *rewinds* the path byopping off the cycle in the end (we note that the cycle may not be simple).

2. **Fast forward operation:** Let \(h_0 = \text{Man}_\epsilon^\tau(\pi_{m-1}) \cdot n_i\). The fast forward condition for a history \(h\) that ends at node \(n_i\) is as follows: there exists a play prefix \(h \cdot \pi'(h)\) consistent with \(\tau\) such that \(n_i \cdot \pi'(h)\) is a proper cycle with average weight less than \(-\epsilon\). In order to be precise, we define \(\pi'(h)\) as the first such prefix according to the lexicographic ordering of the prefixes. If the rewind condition does not hold, and the fast forward condition holds for \(h_0\), then construct \(h_1 = h_0 \cdot \pi'(h_0)\). Continue the process and build \(h_i = h_{i-1} \cdot \pi'(h_{i-1})\), as long as \(h_{i-1}\) satisfies the fast forward condition. If there exists a minimal index \(i \in \mathbb{N}\) such that \(h_i\) does
not satisfy the fast forward condition, then we define \( \text{Man}_\tau^\epsilon(\pi_m) = h_i \). Otherwise, \( \text{Man}_\tau^\epsilon(\pi_m) \) is undefined (not well defined), and we say that the process is stuck in the fast forward operation.

3. **Simulation operation:** Else, if the rewind and the fast forward conditions do not hold, then we have \( \text{Man}_\tau^\epsilon(\pi_m) = \text{Man}_\tau^\epsilon(\pi_{m-1}) \cdot n_i \).

**Intuitive overview of the \( \text{Man}_\tau^\epsilon \) operator.** The \( \text{Man}_\tau^\epsilon \) operator generates an alternative history for the play. The generated history does not contain cycles with average weight more than \(-\epsilon\) (due to the rewind operations) and if the strategy \( \tau \) allows a possible future in which the play returns to the same position and the formed cycle has an average weight less than \(-\epsilon\), then this possible future is added to the history (fast forward operation). The alternative history has the following three key properties:

(i) It is consistent with \( \tau \), i.e., it could really have happened.

(ii) If the average weight of the alternative history is is at least \(-\epsilon\), then the average weight of the actual history is also at least \(-\epsilon\). Hence, if player 1 wins in the play that is induced by the alternative history, then he also wins in the real play.

(iii) When player 1 applies \( \tau \) on the alternative history, all the cycles in the real history have average weight at least \(-\epsilon\). Hence, player 1 does not need to remember the actual cycles in the history (because in the worst case scenario they will simply occur again, and their weights are good for him) and he can play independently of the formed cycles. The next example demonstrates a strategy according to manipulated history.

**Example 8.** Consider the RSM shown in Figure 8.1, and consider a player-1 modular strategy \( \tau \) that follows the edge \( v_1 \rightarrow v_3 \) if \( v_1 \) is visited odd number of times (in the current invocation of \( A_0 \)) and otherwise it follows the edge \( v_1 \rightarrow v_2 \). In this strategy player 1 will play \( v_1 \rightarrow v_3 \) in the first time \( v_1 \) is visited, \( v_1 \rightarrow v_2 \) in the second time, \( v_1 \rightarrow v_3 \) in the third time, and so forth. With this strategy player 1 can ensure a mean-payoff value of at least 0. We now illustrate a play according to the manipulated history for \( \epsilon = \frac{1}{2} \). The play begins by following \( E_n \rightarrow v_1 \) and the \( \text{Man}_\tau^\epsilon \) operator performs a simulation step. So the current manipulated history is \( E_n \rightarrow v_1 \). According to \( \tau \), the next move for player 1 is \( v_1 \rightarrow v_3 \) and if player 2 will then select \( v_3 \rightarrow v_1 \), then a cycle with average weight \(-1 < -\epsilon\) will be formed. Hence, a fast-forward step is made and the manipulated history is now \( E_n \rightarrow v_1 \rightarrow v_3 \rightarrow v_1 \) (the real history is \( E_n \rightarrow v_1 \)). According to the manipulated history, \( v_1 \) has been visited twice, hence the next move for player 1 is \( v_1 \rightarrow v_2 \), and a corresponding simulation step is done for the manipulated history (which is currently \( E_n \rightarrow v_1 \rightarrow v_3 \rightarrow v_1 \rightarrow v_2 \)). We now consider that the next move for player 2 is \( v_2 \rightarrow v_1 \). Hence, the new manipulated history is \( E_n \rightarrow v_1 \rightarrow v_3 \rightarrow v_1 \rightarrow v_2 \rightarrow v_1 \), and since the suffix contains a cycle with average weight \( 1 > -\epsilon \), then a rewind operation is done and the manipulated history is (again) \( E_n \rightarrow v_1 \rightarrow v_3 \rightarrow v_1 \), and therefore the next move for player 1 is (again) \( v_1 \rightarrow v_2 \). We now consider that the next move for player 2 is to invoke the module \( A_0 \). This move is simulated in the manipulated history, and the play continues.
We observe that when playing according to the manipulated history the only real move that player 1 will ever do is $v_1 \rightarrow v_2$, and therefore the obtained strategy is cycle independent (although $\tau$ is not), and it is easy to verify that this strategy ensures a mean-payoff value of at least $-\epsilon$ (and in this example even a positive mean-payoff).

![Figure 8.1: RSM with only one module ($A_0$) and no exit nodes. Player 1 controls the circle vertex and the rest of the vertices are controlled by player 2.](image)

In the following lemma we establish consistency and well-definedness of the manipulated operation for a winning strategy.

**Lemma 71.** Let $\tau$ be a winning strategy (a general winning strategy, not necessarily modular) for the objective $\text{LimInfAvg} \geq 0$. Let $\epsilon > 0$ be an arbitrary constant. We define a strategy $\sigma$ in the following way: for a history $\pi'$ we have $\sigma(\pi') = \tau(\text{Man}_c^\epsilon(\pi'))$. Let $\pi_m$ be a play prefix of length $m$ that is consistent with $\sigma$. Then the following assertions hold:

- $\text{Man}_c^\epsilon(\pi_m)$ is well defined, i.e., the process does not get stuck in the fast forward operation.
- $\text{Man}_c^\epsilon(\pi_m)$ is consistent with $\tau$.

**Proof.** We prove both the items by induction on $m$. In the base case when $m = 0$ (i.e., empty play prefix), all the claims are trivially satisfied. We now consider the inductive case with $m > 0$. Let $\pi_m = \pi_{m-1} \cdot n_i$, for some $n_i \in A_i$, be a play prefix consistent with $\sigma$. By the inductive hypothesis $\text{Man}_c^\epsilon(\pi_{m-1})$ is well defined and consistent with $\tau$. Then $\text{Man}_c^\epsilon(\pi_m)$ is computed by performing one of the following operations.

- **Rewind operation:** In this case clearly $\text{Man}_c^\epsilon(\pi_m)$ is well defined. In addition $\text{Man}_c^\epsilon(\pi_m)$ is a prefix of $\text{Man}_c^\epsilon(\pi_{m-1})$, which is by the inductive hypothesis consistent with $\tau$, hence also $\text{Man}_c^\epsilon(\pi_m)$ is consistent with $\tau$.

- **Fast forward operation:** Towards contradiction, let us assume that the fast forward process enters an infinite loop. We consider the prefix $h_0 = \overrightarrow{h_0} \cdot n_i$, where $\overrightarrow{h_0} = \text{Man}_c^\epsilon(\pi_{m-1})$, and let $\overrightarrow{v_0}$ be the last vertex in $\overrightarrow{h_0}$. The prefix $h_0$ is consistent with $\tau$ for the following reason: $\text{Man}_c^\epsilon(\pi_{m-1})$ is consistent with $\tau$ by the inductive hypothesis, and if $\overrightarrow{v_0}$ is under player 1’s control, then $\tau(\text{Man}_c^\epsilon(\pi_{m-1})) = n_i$ (as
π_m consistent with σ), and otherwise \( \overline{v}_0 \) is under player 2’s control, and since there is a transition from \( \overline{v}_0 \) to \( n_i \) (as \( \pi_m \) is a play prefix) the consistency of \( h_0 \) follows. The prefix \( h_0 \) has an infinite sequence of extensions \( \pi^1, \pi^2, \ldots \) such that the infinite play \( h = h_0\pi^1\pi^2\ldots\pi^j\pi^{j+1}\ldots \) is consistent with \( \tau \) and \( \text{Avg}(\pi^j) < -\epsilon \) for every \( j \in \mathbb{N} \) (by the infinite loop of the fast forward operation). Hence, by definition, \( \liminf \text{Avg}(h) \leq -\epsilon < 0 \). Thus we get that there exists a play consistent with \( \tau \) that is not winning for the objective \( \liminf \text{Avg} \geq 0 \), which contradicts that \( \tau \) is a winning strategy. Hence, the fast forward process always terminates. It follows that \( \text{Man}^\tau(\pi_m) \) is well defined and also by definition of the fast forward operation it is consistent with \( \tau \).

- **Simulation operation**: By definition, \( \text{Man}^\tau(\pi_m) = \text{Man}^\tau(\pi_{m-1}) \cdot n_i \). Let \( \overline{h}_0 = \text{Man}^\tau(\pi_{m-1}) \), and let \( \overline{v}_0 \) be the last vertex in \( \overline{h}_0 \). The prefix \( \text{Man}^\tau(\pi_m) \) is consistent with \( \tau \) for the following reason: \( \text{Man}^\tau(\pi_{m-1}) \) is consistent with \( \tau \) by the inductive hypothesis, and if \( \overline{v}_0 \) is under player 1’s control, then \( \tau(\text{Man}^\tau(\pi_{m-1})) = n_i \), and otherwise \( \overline{v}_0 \) is under player 2’s control, and there is a transition from \( \overline{v}_0 \) to \( n_i \). Thus we have the consistency of \( \text{Man}^\tau(\pi_m) \), and the well-definedness is trivial.

Hence we have that \( \text{Man}^\tau(\pi_m) \) is both consistent with \( \tau \) and well defined. \( \square \)

In the following lemma we obtain a bound on the average of the play prefixes obtained from the manipulated operation of a winning strategy.

**Lemma 72**. Let \( \tau \) be a winning strategy (a general winning strategy, not necessarily modular) for the objective \( \liminf \text{Avg} \geq 0 \). Let \( \epsilon > 0 \) be an arbitrary constant. We define a strategy \( \sigma \) in the following way: for a history \( \pi' \) we have \( \sigma(\pi') = \tau(\text{Man}^\tau(\pi')) \). Let \( \pi_m \) be a play prefix of length \( m \) that is consistent with \( \sigma \). Then we have \( w(\pi_m) \geq w(\text{Man}^\tau(\pi_m)) - \epsilon \cdot |\pi_m| \).

**Proof.** The following claim is the key for the proof.

**Claim.** Every time a rewind operation is done, the cycle \( C \), for which \( \text{Man}^\tau(\pi_m) \cdot C = \text{Man}^\tau(\pi_{m-1}) \cdot n_i \), satisfies that \( \text{Avg}(C) \geq -\epsilon \).

We first prove the claim. Towards a contradiction, assume that \( \text{Avg}(C) < -\epsilon \). Let \( j < m \) be the first index for which \( \text{Man}^\tau(\pi_j) = \text{Man}^\tau(\pi_m) \). Note that such an index must exist (by the definition of rewind operation). We first argue that \( \text{Man}^\tau(\pi_m) \cdot C = \text{Man}^\tau(\pi_{m-1}) \cdot n_i \) is consistent with \( \tau \): (i) \( \text{Man}^\tau(\pi_{m-1}) \) is consistent with \( \tau \) (by Lemma 71); and (ii) let \( \overline{h}_0 = \text{Man}^\tau(\pi_{m-1}) \), and let \( \overline{v}_0 \) be the last vertex in \( \overline{h}_0 \): if \( \overline{v}_0 \) is under player 1’s control, then \( \tau(\text{Man}^\tau(\pi_{m-1})) = n_i \), and otherwise \( \overline{v}_0 \) is under player 2’s control, and there is a transition from \( \overline{v}_0 \) to \( n_i \). Thus we have the consistency of \( \text{Man}^\tau(\pi_m) \cdot C = \text{Man}^\tau(\pi_{m-1}) \cdot n_i \), and it follows that \( \text{Man}^\tau(\pi_j) \cdot C \) is also consistent with \( \tau \). Hence, since \( \text{Avg}(C) < -\epsilon \), a fast forward operation had occurred in position \( j \)

---

\(^1\)only this inequality need not hold for \( \limsup \text{Avg}(h) \)
(note that it is not possible that $\text{Man}^\pi_i(\pi_j)$ was obtained after a rewind operation, since $j$ is the first index for which $\text{Man}^\pi_i(\pi_j) = \text{Man}^\pi_i(\pi_m)$). So by definition, a fast forward operation and not a simulation operation must occur. Hence it is not possible that $\text{Man}^\pi_i(\pi_j) = \text{Man}^\pi_i(\pi_m)$, since at the very least, $\text{Man}^\pi_i(\pi_m) \cdot C$ is a prefix of $\text{Man}^\pi_i(\pi_j)$. Thus, for every $j < m$ we have $\text{Man}^\pi_i(\pi_j) \neq \text{Man}^\pi_i(\pi_m)$, and the contradiction (to the existence of $j$) is obtained.

We now complete the proof of the lemma using the claim. We note that the difference between $\pi_m$ and $\text{Man}^\pi_i(\pi_m)$ contains only (i) cycles with negative weight that were added to $\text{Man}^\pi_i(\pi_m)$ (by fast forward operation) or (ii) cycles with average weight at most $-\epsilon$ and length at most $|\pi_m|$ that were chopped from $\text{Man}^\pi_i(\pi_m)$ (by rewind operation). The desired result follows.

We now show that from a winning strategy for the objective $\text{LimInfAvg} \geq 0$, the strategy obtained using manipulated operation is winning for the objective $\text{LimSupAvg} \geq -2 \cdot \epsilon$.

**Lemma 73.** Let $\tau$ be a winning strategy (a general winning strategy, not necessarily modular) for the objective $\text{LimInfAvg} \geq 0$. Let $\epsilon > 0$ be an arbitrary constant. We define a strategy $\sigma$ in the following way: for a history $\pi'$ we have $\sigma(\pi') = \tau(\text{Man}^\pi_i(\pi'))$. Then $\sigma$ is a winning strategy for the objective $\text{LimSupAvg} \geq -2 \cdot \epsilon$.

**Proof.** Let $\pi$ be a play consistent with $\sigma$, and let $\pi_m$ be the play prefix until position $m$. We consider two cases to complete the proof.

1. In the first case, there exists a constant $n_0 \in \mathbb{N}$ such that for infinitely many indices $m_1, m_2, \ldots$, we have $|\text{Man}^\pi_i(\pi_m)| \leq n_0$. In this case, due to Lemma 72, in positions $m_1, m_2, \ldots$ we get that $w(\pi_m) \geq -n_0 \cdot W - \epsilon \cdot |\pi_m|$. Hence, by definition, $\text{LimSupAvg}(\pi) \geq -\epsilon > -2 \cdot \epsilon$.

2. In the second case, for every $i > 0$ there exists $\ell_i \in \mathbb{N}$, such that for every $m > \ell_i$ we have $|\text{Man}^\pi_i(\pi_m)| \geq i$. By the definition of the manipulation operations, we get that $\text{Man}^\pi_i(\pi_{\ell_i})[0, i] = \text{Man}^\pi_i(\pi_{\ell_i+1})[0, i]$, i.e., the prefix up to length $i$ coincides. Denote $\rho_i = \text{Man}^\pi_i(\pi_{\ell_i})[i]$ the $i$-th position of $\text{Man}^\pi_i(\pi_{\ell_i})$. Due to Lemma 71 the infinite play $\rho = \rho_1 \rho_2 \ldots$ is consistent with $\tau$. Since $\tau$ is a winning strategy we get that $\text{LimInfAvg}(\rho) \geq 0$. Hence there exists infinitely many indices $m_1, m_2, \ldots$ for which $\text{Avg}(\text{Man}^\pi_i(\pi_m)) \geq -\epsilon$, and therefore, due to Lemma 72, we get that $\text{Avg}(\pi_m) \geq -2 \cdot \epsilon$. Hence by definition of $\text{LimSupAvg}$, we obtain $\text{LimSupAvg}(\pi) \geq -2 \cdot \epsilon$.

This concludes the proof of the lemma.

**Lemma 74.** Given a WRG $A$, let $\tau$ be a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$. Let $\epsilon > 0$ be an arbitrary constant. We define a strategy $\sigma$ in the
following way: for a history $\pi'$ we have $\sigma(\pi') = \tau(M_{\pi'}(\pi'))$. Then $\sigma$ is a cycle independent modular strategy.

Proof. In order to verify that $\sigma$ is a modular strategy, we observe that $\text{LocalHistory}(M_{\pi'}(\pi))$ is computable (maybe not effectively) from $\text{LocalHistory}(\pi)$ and that $\tau$ is a modular strategy.

To verify that $\sigma$ is a cycle independent strategy, we observe that if $\pi$ and $\pi \cdot \pi_c$ are consistent with $\sigma$ and $\pi_c$ is a cycle in $\text{CycleFree}(\pi) \cdot \pi_c$, then $M_{\pi'}(\pi \cdot \pi_c) = M_{\pi'}(\pi)$ as $\pi_c$ will be chopped by the rewind operation.

\[\Box\]

Lemma 75. Given a WRG $A$, if there exists a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$ for player 1, then there exists a cycle independent modular winning strategy for the objective for player 1.

Proof. Let $\tau$ be a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$. For every $\epsilon > 0$, define the strategy $\sigma_{\epsilon}$ in the following way: for a history $\pi'$ we have $\sigma_{\epsilon}(\pi') = \tau(M_{\pi'}(\pi'))$. By Lemma 73 and Lemma 74, for every $\epsilon > 0$ we have that $\sigma_{\epsilon}$ is a cycle independent modular winning strategy for the objective $\text{LimSupAvg} > -2 \cdot \epsilon$. There are only a bounded number of cycle independent modular strategies (by Observation 2), thus there is an optimal cycle independent modular strategy, and it must be the case that it is a winning strategy for the $\text{LimSupAvg} \geq 0$ objective (otherwise it does not win for the objective $\text{LimSupAvg} > -\epsilon$ for some $\epsilon > 0$). Let $\sigma$ be that strategy. Let $A^\sigma$ be the player-2 WRG obtained given the strategy $\sigma$. As $\sigma$ is a winning strategy for the objective $\text{LimSupAvg} \geq 0$, then due to Lemma 70 and Lemma 69, the graph $A^\sigma$ does not have a negative non-decreasing cycle. Hence due to Lemma 70 and Lemma 69, the strategy $\sigma$ is winning also for the objective $\text{LimInfAvg} \geq 0$. This completes the proof of the result.

\[\Box\]

The next theorem is an immediate consequence of Lemma 75.

Theorem 18. Given a WRG $A$, the problem of deciding if player 1 has a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$ is decidable.

Proof. By Lemma 75 it is enough to check if player 1 has a cycle independent modular strategy. As the number of such strategies is bounded (by Observation 2), it is enough to construct the graph $A^\tau$ for every cycle independent modular strategy $\tau$ and check if in $A^\tau$ there exists a path $\pi$ with $\text{LimInfAvg}(\pi) < 0$ (this check is achieved using the algorithms of Section 7.1).

\[\Box\]

Objective $\text{LimSupAvg} \geq 0$. The proof of Lemma 75 shows that if player 1 has a cycle independent modular winning strategy for the $\text{LimSupAvg} \geq 0$ objective, then he also
has a modular winning strategy for the \( \text{LimInfAvg} \geq 0 \) objective. However, it does not imply that an arbitrary modular winning strategy for the \( \text{LimSupAvg} \geq 0 \) objective can be transformed into a winning strategy for the \( \text{LimInfAvg} \geq 0 \) objective, or that it can be transformed into a cycle independent modular strategy. The proof for the objective \( \text{LimSupAvg} \geq 0 \) will reuse many parts of the proof for the objective \( \text{LimInfAvg} \geq 0 \), however, some parts of the proof are different and we present them below. In fact we show that for modular winning strategies the objective \( \text{LimSupAvg} \geq 0 \) coincides with the objective \( \text{LimInfAvg} \geq 0 \). For the proof we need the notion of non-negative cycle free local history, which we define below.

**Non-negative cycle free local history operator.** Consider a WRG \( \mathcal{A} \) and let \( \tau \) be a modular strategy, and \( \pi \) be a path in \( \mathcal{A} \), consistent with \( \tau \), that begins at the entry of module \( A_i \) and ends at module \( A_i \) (in the same stack height). The **non-negative cycle free local history operator** is defined as follows:

1. For \(|\text{LocalHistory}(\pi)| = 1\) we have \( \text{NonNegCFLocalHistory}^\tau(\pi) = \text{LocalHistory}(\pi) \).

2. For \(|\text{LocalHistory}(\pi)| > 1\), let \( \pi = \pi_0v_i \) such that \( \text{NonNegCFLocalHistory}^\tau(\pi_0) = u_0u_1\ldots u_m \). Let \( j \in \{0, \ldots, m\} \) be the first index such that \( u_j = v_i \), and every sub-play \( \pi^* \) consistent with \( \tau \) with local history \( u_ju_{j+1}\ldots u_mv_i \) is a cycle with non-negative total weight. If such index \( j \) exists, then \( \text{NonNegCFLocalHistory}^\tau(\pi) = u_0\ldots u_j \), otherwise \( \text{NonNegCFLocalHistory}^\tau(\pi) = u_0u_1\ldots u_mv_i \).

Informally, the \( \text{NonNegCFLocalHistory}^\tau \) operator removes cycles that are ensured to have non-negative total weight from the local history.

**Non-negative cycle independent modular strategy.** Given a modular strategy \( \tau \), the **non-negative cycle independent modular strategy** \( \sigma \) of \( \tau \), is defined as follows: For a local history \( \rho \) we have \( \sigma(\rho) = \tau(\text{NonNegCFLocalHistory}^\tau(\rho)) \). We now present some notations required for the proofs.

**Sure non-negative cycle and proper simple cycle.** Given a modular strategy \( \tau \), a path \( \pi = v_0v_1\ldots v_m \) is a **sure non-negative cycle** if \( \pi \) is a proper cycle consistent with \( \tau \), and every path \( \pi' \) consistent with \( \tau \) such that \( \text{LocalHistory}(\pi) = \text{LocalHistory}(\pi') \) is a proper cycle with non-negative weight. A path \( \pi \) is a **proper simple cycle** if \( \pi \) is a proper cycle, and \( \text{LocalHistory}(\pi) \) is a simple cycle.

**Lemma 76.** If \( \tau \) is a modular winning strategy for the objective \( \text{LimSupAvg} \geq 0 \), then the **non-negative cycle independent modular strategy** \( \sigma \) of \( \tau \) is also a winning strategy for the objective \( \text{LimSupAvg} \geq 0 \).

**Proof.** The proof is essentially similar to the proof of the result for the objective \( \text{LimInfAvg} \geq 0 \), and we present a succinct argument (as it is very similar to the previous proofs for \( \text{LimInfAvg} \geq 0 \)). As previously, we define the manipulated operations on histories such that every sure non-negative cycle is chopped (by the rewind operation) from the history. By definition, the non-negative cycle independent strategy \( \sigma \) of \( \tau \)
makes choices according to the choices of \( \tau \) on the manipulated history. The weight of the manipulated history, in every position, is at most the weight of the original history, since only non-negative cycles are chopped. By similar arguments to those presented in Lemma 73, we get that the \( \text{LimSupAvg} \) of the original history is at most 0, and thus the non-negative cycle independent strategy \( \sigma \) of \( \tau \) is a winning strategy.

In the following lemmas we establish that for modular strategies the objectives \( \text{LimSupAvg} \geq 0 \) and \( \text{LimInfAvg} \geq 0 \) coincide.

Lemma 77. Given a WRG \( A \), let \( \sigma \) be a non-negative cycle independent modular strategy. Let \( A_i \) be a module in \( A \). Then there exist \( n_\sigma \in \mathbb{N} \) and \( \delta_\sigma > 0 \), such that for every possible non-negative cycle free local histories \( h_0 \) and \( h_1 \) of the module \( A_i \), and for every sub-play \( \rho \), consistent with \( \sigma \) such that

- \( \rho \) is a proper cycle with negative weight; and
- the non-negative cycle free local history of \( A_i \) before (resp. after) \( \rho \) was played is \( h_0 \) (resp. \( h_1 \));

there exists a sub-play \( \rho_{h_0,h_1} \), consistent with \( \sigma \) and that satisfies the items above, such that every play prefix of \( \rho_{h_0,h_1} \) with length at least \( n_\sigma \) has an average weight of at most \( -\delta_\sigma \).

Proof. Let \((b_1, n_1), \ldots, (b_m, n_m)\) be the pairs of all boxes and their return nodes that appear in module \( A_i \). For every pair \((b_j, n_j)\), let \( \pi_{b_j,n_j} \) be one of the shortest plays, consistent with \( \sigma \), with minimal weight from \( b_j \) to \( n_j \). If such a path exists we denote \( w((b_j, n_j)) = w(\pi_{b_j,n_j}) \). Let \( X \) be the maximal such weight. W.l.o.g all the weights of the edges that occur in \( A_i \) are at most \( X \), and \( X \geq 0 \).

For every \( b_j, n_j \) such that there exists a play from \( b_j \) to \( n_j \) consistent with \( \sigma \), but a minimal-weight play does not exist, we denote by \( \pi_{b_j,n_j} \) one of the shortest plays consistent with \( \sigma \), with minimal weight from \( b_j \) to \( n_j \). If such a path exists we denote \( w((b_j, n_j)) = w(\pi_{b_j,n_j}) \). Let \( X \) be the maximal such weight. W.l.o.g all the weights of the edges that occur in \( A_i \) are at most \( X \), and \( X \geq 0 \).

First, we form \( \rho' \) from \( \rho \) by removing all the sure non-negative cycles from \( \rho \). Clearly, (i) the sum of weights of \( \rho' \) is negative; and (ii) \( \rho' \) is consistent with \( \sigma \), because \( \rho \) is consistent with \( \sigma \) and \( \sigma \) is a non-negative cycle independent strategy. Moreover, the non-negative cycle free local history after playing \( \rho' \) is the same as after playing \( \rho \). Next we form \( \rho'' \) from \( \rho' \) by replacing every sub-play from \( b_j \) to \( n_j \) (such that \( n_j \) is the first time the sub-play enters \( A_i \)) in \( \rho' \) with \( \pi_{b_j,n_j} \). Again, \( \rho'' \) is consistent with \( \sigma \), since \( \sigma \) is a modular strategy. In addition, the local history of \( A_i \) was not changed at all.

We claim that \( \rho'' \) does not contain simple proper non-negative cycles in module \( A_i \). Indeed, towards a contradiction let \( v_1v_2 \ldots v_m \) be the local history of the first such
cycle (note that $m \leq |V_i|$). Note that if the cycle contains a sub-play $\pi_{b_j,n_j}$ such that $w(\pi_{b_j,n_j}) = -20 \cdot |V_i| \cdot X$, then the sum of the weights of the cycle cannot be non-negative. Hence it follows that this cycle is the cycle with minimal weight among all simple cycles with local history $v_1 v_2 \ldots v_m$. Hence this cycle is sure non-negative, which contradicts the fact that $v_1 v_2 \ldots v_m$ is a local history of sub-play of $\rho''$. Thus for every simple proper cycle in $\rho''$ the sum of the weights of the cycle is negative. Moreover, the length of every simple proper cycle in $\rho''$ is at most $|V_i| \cdot |\Pi|$. Hence every sub-play of $\rho''$ with length at least $n_\sigma = (|V_i| \cdot |\Pi| \cdot X)^2$ will have an average weight of at most $-\delta_\sigma = -\frac{1}{|V_i| \cdot |\Pi| \cdot X}$. Note that $n_\sigma$ and $\delta_\sigma$ do not depend on $\rho$, and hence the desired result follows.

Lemma 78. Let $\sigma$ be a non-negative cycle independent modular strategy. If there exists a play $\rho$ consistent with $\sigma$ such that the suffix of $\rho$ is an infinite sequence of proper cycles $C_1, C_2, \ldots$ with negative weights, then $\sigma$ is not a winning strategy for the objective $\text{LimSupAvg} \geq 0$.

Proof. Let $\rho = \rho_0 C_1 C_2 \ldots C_i \ldots$, and let $A_i$ and $n_i$ be the module and the vertex, respectively, that all the cycles begin and end in. Let $n_\sigma$, $\delta_\sigma$ be the constants from Lemma 77. Let $h_0^i$ be the non-negative cycle free local history of $A_i$ before cycle $C_i$ is played, and $h_1^i$ be the non-negative cycle free local history of $A_i$ after cycle $C_i$ is played. By Lemma 77, for every cycle $C_i$ there exists a cycle $C_{h_0^i,h_1^i}$ consistent with $\sigma$, with negative weight, and the average weight of every sub-play of $C_{h_0^i,h_1^i}$ longer than $n_\sigma$ is at most $-\delta_\sigma$. The play $\rho' = \rho_0 C_1 C_2 \ldots C_{i-1} C_{h_0^i,h_1^i} C_{i+1} \ldots$ is consistent with $\sigma$, since $C_{h_0^i,h_1^i}$ is consistent with the initial non-negative cycle free local history $h_0^i$, and the non-negative cycle free local history after playing $C_{h_0^i,h_1^i}$ is $h_1^i$. Since $\sigma$ is a non-negative cycle independent strategy, the play $\rho^* = \rho_0 C_{h_0^0,h_1^0} C_{h_0^1,h_1^1} \ldots C_{h_0^n,h_1^n} \ldots$ is consistent with $\sigma$. On the other hand, it is straightforward to verify that we have $\text{LimSupAvg}(\rho^*) \leq \max\{-\frac{1}{n_\sigma},-\delta_\sigma\} < 0$. Hence $\sigma$ is not a winning strategy for the objective $\text{LimSupAvg} \geq 0$, and the desired result follows.

Lemma 79. If player 1 has a modular winning strategy for the objective $\text{LimSupAvg} \geq 0$, then for every $\epsilon > 0$, player 1 has a cycle independent modular winning strategy $\sigma$ for the objective $\text{LimSupAvg} \geq -\epsilon$.

Proof. Let $\tau^*$ be a modular winning strategy for the objective $\text{LimSupAvg} \geq 0$, and let $\tau$ be the non-negative cycle independent strategy of $\tau^*$. For $\epsilon > 0$, consider the cycle independent modular strategy $\sigma$ such that for histories $\pi$ we have $\sigma(\pi) = \text{Man}_\epsilon(\pi)$. By Lemma 77 the strategy $\tau$ is also a winning strategy. We note that all the arguments for the objective $\text{LimInfAvg} \geq 0$ also hold for the objective $\text{LimSupAvg} \geq 0$, other than the inequality (mentioned as footnote) in Lemma 71. We replace the inequality (mentioned as footnote) of Lemma 71 with Lemma 78, and repeat exactly the same arguments for the objective $\text{LimInfAvg} \geq 0$ (up to Lemma 75) to obtain the desired result.
We obtain the following result as a corollary, and all the desired results follow for the objective $\text{LimSupAvg} \geq 0$.

**Corollary 4.** Given a WRG $A$, there exists a modular winning strategy for the objective $\text{LimSupAvg} \geq 0$ iff there exists a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$.

### 8.1.2 Modular winning strategy problem in NP

In this section we will show that the modular winning strategy problem is in NP. To show the result we introduce the notion of a signature game.

**The signature game.** Let $G = ((V, E), (V_1, V_2))$ be a finite two-player game graph (on finite directed graph $(V, E)$) with vertex set $V$, edge set $E$, and partition $(V_1, V_2)$ of the vertex set into player-1 (resp. player-2) vertex set $V_1$ (resp. $V_2$). Let the game graph be equipped with a weight function $w : E \rightarrow \mathbb{Z} \cup \{-\omega\}$. Let the initial vertex be $v_0 \in V$. Let $\vec{\nu} = (\nu_1, \ldots, \nu_{|V|})$ be a threshold vector such that all $\nu_i \in \mathbb{Z} \cup \{+\infty, -\omega\}$.

The weight of a finite path in $G$ is the sum of the edge weights of the path, according to the following convention: $-\omega + z = -\omega$, for any $z \in \mathbb{Z} \cup \{-\omega\}$. A signature game consists of a tuple $(G, \vec{\nu})$, where $G$ is a two-player game graph, and $\vec{\nu}$ is the threshold vector. For a play $\rho = \rho_0 \rho_1 \rho_2 \ldots \rho_j \rho_{j+1} \ldots$, player 1 is the winner if both the following two conditions hold:

- The play $\rho$ does not contain a negative cycle, or a cycle that has an edge with weight $-\omega$.
- For every $j \in \mathbb{N}$, if $\rho_j = v_i$ (i.e., the $j$-th index of the play is the $i$-th vertex), then $w(\rho_0 \rho_1 \rho_2 \ldots \rho_j) \geq \nu_i$, according to the convention (i) $-\omega < z$ for every $z \in \mathbb{Z}$, and (ii) $z, -\omega < +\infty$ for every $z \in \mathbb{Z}$. In other words, the sum of the weights upto any index $j$ (with the $i$-th vertex in index $j$) must be at least $\nu_i$, and no $-\omega$-edge must be visited unless $\nu_i = -\omega$.

We will consider signature games such that if $\nu_i \in \mathbb{Z}$, then $\nu_i \geq -2 \cdot W \cdot |V|$, where $W$ is the maximum absolute values of the integer weights in graph $G$. If for a vertex $v$, the threshold value is $+\infty$, then to ensure winning player 1 must ensure that $v$ is never visited. In other words, vertices with $+\infty$ threshold must be avoided (it can be interpreted as a safety objective with $+\infty$ vertices as non-safe vertices to be avoided).

Our first goal is to show that memoryless winning strategies exist in signature games, and the result will be obtained by a reduction to finite-state mean-payoff games. Given a signature game $(G, \vec{\nu})$ we define an auxiliary finite-state mean-payoff game as follows.

**From signature game $(G, \vec{\nu})$ to auxiliary mean-payoff game $\overline{G}_{\vec{\nu}}$.** Given a signature game $(G, \vec{\nu})$ we construct a finite-state auxiliary mean-payoff game $\overline{G}_{\vec{\nu}} = ((\overline{V}, \overline{E}), (\overline{V}_1, \overline{V}_2))$, with a weight function $\overline{w}$ as follows: let the signature game graph be $G = ((V, E), (V_1, V_2))$ and the weight function in $G$ be $w_G$. Then we have the following components in the auxiliary game:
• (Vertex set and partition). $\overline{V} = V \times \{1, 2\}$; and $\overline{V}_1 = V_1 \times \{1\}$.

• (Edges). $\overline{E} = \{(u, 1), (v, 2)\} \cup \{(v, 2), (v, 1)\} \cup \{(v_i, 2), (v_0, 1)\} | v_i \in V, v_i \neq \omega\}.$

• (Weight function). If $w_G(u, v) \neq -\omega$, then $w((u, 1), (v, 2)) = w_G(u, v)$; otherwise (we have $w_G(u, v) = -\omega$) we set $w((u, 1), (v, 2)) = -10 \cdot W \cdot |V|$. Moreover, $w((v_i, 2), (v_0, 1)) = -\nu_i$ and all the other edges are assigned with zero weight. Note that if $\nu_i = +\infty$, then $w$ assigns weight $-\infty$, and to win player 1 must avoid such edges (can be interpreted as a safety condition).

Informally, the auxiliary mean-payoff game is constructed from the signature game by adding for every vertex $v_i$ a fresh copy (vertex $(v_i, 2)$ and the original vertex is represented as $(v_i, 1)$), and an option for player 2 to return to the initial vertex $(v_0, 1)$ “paying” cost $-\nu_i$ (whenever $\nu_i \neq -\omega$). Thus, if at any position of the play the current vertex is $(v_i, 2)$, and the sum of the weights since the last visit to $(v_0, 1)$ is less than $\nu_i$, then player 2 can ensure that a negative cycle is completed. The mean-payoff objective of player 1 is to ensure non-negative average payoff. Also note that player 1 must avoid the $-\infty$ edge weights, and equivalently it can be treated as a mean-payoff safety game.

In the following lemma we establish the relation of the signature game and the auxiliary game.

Lemma 80. Let $(G, \vec{\nu})$ be a signature game such that $\nu_i \geq -2 \cdot W \cdot |V|$ for every $\nu_i \in \mathbb{Z}$, and let $\overline{G}_\nu$ be the corresponding auxiliary mean-payoff game. Then the following statements are equivalent:

1. Player 1 is the winner in the auxiliary mean-payoff game $\overline{G}_\nu$ (i.e., player 1 can ensure non-negative mean-payoff).

2. Player 1 has a memoryless winning strategy in the signature game $(G, \vec{\nu})$.

3. Player 1 is the winner in the signature game $(G, \vec{\nu})$.

Proof. We first prove that item 1 implies item 2.

1. In order to prove that item 1 implies item 2, let us assume that player 1 is the winner in the auxiliary mean-payoff game. Note that the auxiliary mean-payoff game is equivalently a finite-state mean-payoff safety game, and therefore player 1 has a memoryless winning strategy $\tau$ in the auxiliary mean-payoff game [47] (a mean-payoff safety game is easily transformed to a mean-payoff game by making the non-safe vertices absorbing with negative weights). Hence the memoryless strategy $\tau$ ensures that edges with weight $-\infty$ are never visited. Towards a contradiction, let us assume that $\tau$ is not a winning strategy for the signature game (note that $\tau$ is also a well-defined player-1 strategy in the signature game choosing edges in copy 1 according to $\tau$). Therefore one of the following two cases occur.
Case 1: There exists a finite play prefix $\rho$ that is consistent with $\tau$, which starts from the initial vertex $v_0$ to some vertex $v_i \in V$ with sum of weights less than $\nu_i$. In this case, either $\rho$ goes through an $-\omega$ edge, or $w_G(\rho) < \nu_i$. If $\rho$ goes through an $-\omega$ edge in $G$, then the weight of $\rho$ in $G_\vec{\nu}$ is at most $-9 \cdot W \cdot |V|$, since w.l.o.g we can assume that $\rho$ does not have positive cycles (as $\tau$ is memoryless). As $-9 \cdot W \cdot |V| < \nu_i$, it follows that the path $(\rho \cdot ((v_i, 2), (v_0, 1)))^\omega$ is consistent with $\tau$ and has a negative mean-payoff in the auxiliary game. This contradicts the assumption that $\tau$ is a winning strategy. If $\rho$ does not go through an $-\omega$ edge, then $w_G(\rho) < \nu_i$ and again $(\rho \cdot ((v_i, 2), (v_0, 1)))^\omega$ is consistent with $\tau$ and has a negative mean-payoff in the auxiliary game. This is again a contradiction that $\tau$ is a winning strategy, and concludes the proof of the first case.

Case 2: There exists a finite play prefix $\rho = \rho_1 \cdot \rho_2$ that is consistent with $\tau$, such that $\rho_2$ is a negative cycle (or a cycle with $-\omega$ edge) in the signature game graph. If $\rho_2$ does not contain an $-\omega$ edge $e$, then by definition, $\rho_2$ is a negative cycle also in the auxiliary game. Otherwise, $\rho_2$ contains an $-\omega$ edge $e$, and then again $\rho_2$ is a negative cycle in the auxiliary game, as w.l.o.g we can assume that $\rho_2$ does not contain positive cycles, and since $\pi(e) \leq -10 \cdot W \cdot |V|$. Thus the play $\rho_1 \cdot (\rho_2)^\omega$ is consistent with $\tau$ and has a negative mean-payoff in the auxiliary game. This contradicts the assumption that $\tau$ is a winning strategy, and completes the proof.

2. Item 2 trivially implies item 3.

3. We now show that item 3 immediately implies item 1. It is straightforward to verify that if player 1 plays according to the signature game winning strategy in every position, then a negative cycle will not be formed in the auxiliary game (as a negative cycle is not formed in the signature game, and the threshold vector is always satisfied in the signature game) and a vertex with threshold $+\infty$ will never be reached. Hence the mean-payoff of the play in the auxiliary mean-payoff game will be non-negative and $-\infty$ edges will never be visited. This shows that item 3 implies item 1.

This completes the proof.

Lemma 81. Let $(G, \vec{\nu})$ be a signature game such that $\nu_i \geq -2 \cdot W \cdot |V|$ for every $\nu_i \in \mathbb{Z}$. There is a winning strategy for player 1 in the signature game $(G, \vec{\nu})$ iff player 1 has a memoryless winning strategy.

Proof. Follows from Lemma 80.
Remark 3. The result of Lemma 81 holds for all thresholds $\nu_i$, but we consider $\nu_i \geq -2 \cdot W \cdot |V|$ for simplicity of the proof.

Signature games to memoryless modular strategies. We will now use the existence of cycle independent modular winning strategies, and memoryless winning strategies in signature games to show the existence of memoryless modular strategies. For simplicity we will consider recursive game graphs where every module has a single entry, and a simple polynomial reduction from multi-entry recursive game graphs to single entry recursive game graphs is established in [9]. To prove the result of memoryless modular strategies we define the signature games for modular strategies.

Signature games for modular strategies. Consider a WRG $A = \{A_1, A_2, \ldots, A_n\}$ and let $\tau = \{\tau_i\}_{i=1}^n$ be a modular strategy. Consider a module $A_i$ in $A$. Let $b \in B_i$ be a box in module $A_i$, which invokes the module $A_j$, and let $n_i \in N_i$ be a node in module $A_i$ that is connected to one exit of $A_j$, which is reachable according to the strategy $\tau$. We denote by $w^\tau_{b,n_i}$ the minimal weight of all plays according to $\tau$ that begins at the call to box $b$ and ends at $n_i$ (in the same stack height), and do not visit any other vertices in $A_i$ (in the same stack height). If such a minimal-weight play does not exist, then let $w^\tau_{b,n_i} = -\omega$. For every module $A_i$, we form a finite-state two-player game graph $G_{A_i}$, with a weight function as follows: (i) in the module $A_i$ we add an edge from every box $b$ to every return node $n_i$ with weight $w^\tau_{b,n_i}$, and add a self loop, with weight 0, to every exit node; and (ii) every box is now interpreted as a player-2 vertex. Note that the local strategy $\tau_i$ is a well-defined player-1 strategy in the game graph $G_{A_i}$. For a vertex $v \in V_i$, (i) if $v$ is visited along a play consistent with $\tau_i$, then let $\eta_v$ denote the maximal value such that in every position of a play according to $\tau_i$ on $G_{A_i}$, that begins in the entry node of $A_i$, and is currently at vertex $v \in V_i$, the sum of weights from the beginning of the play is at least $\eta_v$; and (ii) otherwise, $v$ is never visited along all plays consistent with $\tau_i$, then $\eta_v = +\infty$ (note that this is like a safety condition to ensure that $v$ is not visited). The signature game for $\tau$ on module $A_i$ consists of the game graph $G_{A_i}$ and the threshold vector $\bar{\nu} \in \{-\omega, +\infty\} \cup \mathbb{Z}^{|V_i|}$, such that $\nu^i_v = \eta_v$ for all vertices $v \in V_i$. We denote by $(G_{A_i}, \bar{\nu}, \tau)$ the signature game obtained given the modular strategy $\tau$ on module $A_i$. We first establish some basic properties in Lemma 82, and then in Lemma 83 we establish properties of the winning strategies in the signature games from modular strategies.

Lemma 82. Let $A$ be a WRG. If $\tau$ is a cycle independent modular winning strategy for the objective LimInfAvg $\geq 0$, then for every module $A_i$ the following assertions hold:

- $\tau_i$ is a winning strategy for the signature game $(G_{A_i}, \bar{\nu}, \tau)$; and

- the integer coefficients of $\bar{\nu}^i$ are at least $-2 \cdot W \cdot |V_i|$. 

Proof. The first fact of the lemma follows from the facts that every path according to $\tau_i$ has sum of weights at least $\bar{\nu}$ and does not contain negative cycles (by the definition of the signature game given $\tau$, since $\tau$ is a winning strategy). The second fact of the
Lemma 83. Let $A$ be a WRG. Let $\tau = \{\tau_i\}_{i=1}^n$ be a cycle independent modular winning strategy in $A$ for the objective $\text{LimInfAvg} \geq 0$. Let $\sigma = \{\sigma_i\}_{i=1}^n$ be a modular strategy such that $\sigma_i$ is a winning strategy for the signature game $(G_{A_i}, \vec{\nu}_i, \tau)$. Then for every play $\rho^\sigma$, consistent with $\sigma$, which starts from the entry node of a module $A_i$ to a node $n$ in the same module (and possibly goes through box nodes), there exists a play $\rho^\tau$, consistent with $\tau$, from the same entry node to the same node $n$, such that $w(\rho^\sigma) \geq w(\rho^\tau)$. In addition, the path $\rho^\sigma$ does not contain a negative proper cycle.

Proof. The proof is by induction on the additional stack height of $\rho^\sigma$.

- **Base case:** Additional stack height is 0. In this case the play $\rho^\sigma$ have only edges from $A_i$, and the weight of the play is identical to the weight of the same play in $(G_{A_i}, \vec{\nu}_i, \tau)$. The play $\rho^\sigma$ does not visit a vertex with threshold $+\infty$, otherwise $\sigma_i$ would not be a winning strategy in the signature game $(G_{A_i}, \vec{\nu}_i, \tau)$. Hence, by definition, there exists a play $\rho^\tau$, consistent with $\tau$ from the entry node to $n$ with weight at most $\nu^i_n$. Since $\sigma_i$ is a winning strategy in the signature game, and $\rho^\sigma$ is consistent with $\sigma_i$, we get that $w(\rho^\sigma) \geq \nu^i_n$. Therefore $w(\rho^\sigma) \geq w(\rho^\tau)$. Since $\sigma_i$ is a winning strategy in the signature game, and the path $\rho^\sigma$ is consistent with $\sigma_i$ also in graph $G_{A_i}$, we get that $\rho^\sigma$ does not contain negative cycles.

- **Inductive step:** Additional stack height $> 0$. For simplicity, we first assume that $\rho^\sigma$ goes only through one box node in the module $A_i$ (in the first stack level). Let node $b$ be that box, and let node $u \in A_i$ be the return node in that path. Let $\rho^\sigma_{b,u}$ be the sub-play from the entry node of $b$ to node $u$. Recall that $w^\tau_{b,u}$ is the minimal weight among all plays consistent with $\tau$ between $b$ and $u$. Let $A_j$ be the module invoked by $b$, and let $u'$ be the exit node that leads to the return node $u$ in $A_i$. As the additional stack height from the entry node of $A_j$ to $u'$ is strictly smaller than the additional stack height of $\rho^\sigma$, it follows from the inductive hypothesis that there exists a path consistent with $\tau$ between these two nodes with weight at most $w(\rho^\sigma_{b,u})$. Hence $w^\tau_{b,u} \leq w(\rho^\sigma_{b,u})$. Thus, the weight of $\rho^\sigma$ is bounded from below by the induced path of $\rho^\sigma$ over the signature game $(G_{A_i}, \vec{\nu}_i, \tau)$. Thus, by the definition of the signature game there exists a path $\rho^\tau$ as desired. In addition, by the inductive hypothesis, the path $\rho^\sigma_{b,u}$ does not contain proper negative cycles, and by the same arguments as above, there is also no negative proper cycle in module $A_i$. The case where $\rho^\sigma$ goes through more than one box, is a straightforward extension of the argument presented above.

Thus we have the desired result. □
Lemma 84. Let \( \mathcal{A} \) be a WRG. Let \( \tau = \{\tau_i\}_{i=1}^n \) be a cycle independent modular winning strategy in \( \mathcal{A} \) for the objective \( \text{LimInfAvg} \geq 0 \). Let \( \sigma = \{\sigma_i\}_{i=1}^n \) be a memoryless modular strategy such that \( \sigma_i \) is a memoryless winning strategy for the signature game \((G_A, \vec{\nu}, \tau)\). Then \( \sigma \) is a memoryless modular winning strategy in \( \mathcal{A} \) for the objective \( \text{LimInfAvg} \geq 0 \).

Proof. Let \( \mathcal{A}^\sigma \) be the player-2 WRG obtained by fixing the memoryless modular strategy \( \sigma \) in \( \mathcal{A} \). Assume towards a contradiction that \( \mathcal{A}^\sigma \) has a reachable non-decreasing negative cycle \( C \), and let \( \rho \) be a finite path that leads to the first vertex of \( C \). By Lemma 83 it follows that \( C \) cannot be a proper cycle.

First, we argue that in \( \mathcal{A} \) there exists a finite path \( \rho^\tau \) from the first vertex of \( \rho \) to the last vertex of \( \rho \) that is consistent with \( \tau \). Indeed, by Lemma 83, between every entry node and box node in \( \rho \) there exists a path consistent with \( \tau \), and finally there also exists such a path between the last entry node and the last node of \( \rho \).

Second, to achieve the contradiction we will show that \( \tau \) is not a winning strategy. Let \( e_1 \) be the first entry node in \( C \) (it must exist as \( C \) is not a proper cycle), and \( n_i \) be the last (and the first) node in \( C \) (note that \( C \) is not a proper cycle, and hence this node is well defined). Note that for every \( m \in \mathbb{N} \), the path \( C^m \) is a non-decreasing cycle that is consistent with \( \sigma \) (as \( \sigma \) is a memoryless strategy). Let \( \rho_{e_1,n_i} \) be the path from \( e_1 \) to \( n_i \). Let \( \rho_{n_i,e_1} \) be the path from \( n_i \) to first appearance of \( e_1 \) in \( C \). We consider the path \( \rho^* = \rho_{e_1,n_i} \cdot C^m \cdot \rho_{n_i,e_1} \), for \( m = 2 \cdot W \cdot (|\rho_{e_1,n_i}| + |\rho_{n_i,e_1}|) \). This is a path that is (i) consistent with \( \sigma \), (ii) begins and ends in the entry node \( e_1 \) of the same module (not necessarily in the same stack height). Let \( b_1, b_2, \ldots, b_\ell \) be the boxes that occur in the path. By Lemma 83, for every \( k \) there exists a path \( \rho_{b_k,b_{k+1}}^\tau \) consistent with \( \tau \) such that \( w(\rho_{b_k,b_{k+1}}^\tau) \leq w(\rho_{b_k,b_{k+1}}) \). Hence there exists a path \( \rho^\tau_i \) that is consistent with \( \tau \) from \( e_1 \) to \( e_1 \) such that the sum of the weights is negative. As \( \tau \) is a modular strategy, and \( e_1 \) is an entry node, it follows that the path \( \rho^\tau_i \) is also consistent with \( \tau \), and has a negative mean-payoff. In conclusion, we obtain that there exists a reachable negative non-decreasing cycle in \( \mathcal{A} \) consistent with \( \tau \), and this contradicts that \( \tau \) is a winning strategy.

Hence, every path consistent with \( \sigma \) does not contain a negative non-decreasing cycle. By Lemma 69 it follows that \( \sigma \) is a winning strategy in \( \mathcal{A} \) for the objective \( \text{LimInfAvg} \geq 0 \).

The following example illustrates the connection between signature games and cycle independent modular winning strategies.

Example 9. Consider the RSM \( \langle A_0, A_1 \rangle \) (shown in Figure 8.2) and a player-1 modular strategy \( \tau = \{\tau_0, \tau_1\} \) such that in module \( A_0 \) the strategy \( \tau_0 \) always selects \( v_4 \to v_5 \) if the play visited \( v_3 \) and otherwise it always invokes \( A_1 \), and in module \( A_1 \), the strategy \( \tau_1 \) selects the upper exit (denoted by \( \text{Ex}_1 \)) if \( v_3 \) is visited (in the current invocation of \( A_1 \)) and otherwise it selects the lower exit (denoted by \( \text{Ex}_2 \)). The strategy \( \tau \) is a cycle.
independent strategy, but it is not a memoryless strategy. In a play according to $\tau$ if the upper exit of $A_1$ is reached, then the path $E_{u_1} \to u_1 \to u_3 \to E_{x_2}$ with weight $-1$ is played, and if the lower exit is reached, then the weight of the sub-play is 1. Hence, if in module $A_0$, vertex $v_4$ invokes $A_1$, then if the upper exit of $A_1$ is reached, then the play continues to $v_5$ and from there to $v_4$ and a cycle with weight 0 is formed. If the lower exit of $A_1$ is reached, then the play continues to $v_4$ and a cycle with weight 1 is formed. If in $v_4$ the player-1 move is $v_4 \to v_5$, then the play continues to $v_4$, and a cycle with weight 0 is formed. Therefore, the strategy $\tau$ ensure that mean-payoff is at least 0.

The corresponding signature game is illustrated in Figure 8.3. Note that the box that invokes $A_1$ is replaced by $b_{A_1}$. We note that player 1 has to decide on the next move only in $u_4$ in $A_1$ and in $v_4$ in $A_0$. Hence, the strategies $\tau_0$ and $\tau_1$ are well defined over $G_{A_0}$ and $G_{A_1}$, respectively. The strategy $\tau_1$ ensures the following signature over $G_{A_1}$: $\nu^0 = 0, \nu^1 = 0, \nu^2 = 7, \nu^3 = 6, \nu^4 = 6, \nu^5 = -1, \nu^6 = 1$ and the strategy $\tau_0$ ensures the following signature over $G_{A_0}$: $\nu^0 = 0, \nu^1 = 0, \nu^2 = 1, \nu^3 = -1, \nu^4 = 2, \nu^5 = 7, \nu^{b_{A_1}} = 2, \nu^6 = 1, \nu^7 = 3$. The same signatures are satisfied by a memoryless strategy that always selects $u_4 \to v_5$ in $G_{A_1}$ and $v_4 \to b_{A_1}$ in $G_{A_0}$. The corresponding memoryless strategy in $A_1$ is to select the upper exit in $u_4$ and in $A_0$ is to invoke $A_1$ when in $v_4$. It is easy to verify that this memoryless strategy is a winning strategy in the recursive game for the mean-payoff objective.

Figure 8.2: RSM with two modules ($A_0$ and $A_1$). Player 1 controls the round vertices and the rest of the vertices are controlled by player 2.
Chapter 8. Pushdown Mean-Payoff Games with Modular Strategies

Lemma 85. Let $A$ be a WRG. Player 1 has a modular winning strategy for the objective $\text{LimInfAvg} \geq 0$ iff there exists a memoryless modular winning strategy for player 1 for the $\text{LimInfAvg} \geq 0$ objective.

Proof. The proof for the direction from right to left is trivial. The opposite direction is obtained as follows: by Lemma 75 it follows that if there is a modular winning strategy, then there is a cycle independent modular winning strategy; and by Lemma 84 it follows that if there is a cycle independent modular winning strategy, then there is a memoryless modular winning strategy. The desired result follows.

We are now ready to prove the main result of this section.

Theorem 19. The problem of deciding if player 1 has a modular winning strategy in a WRG $A$ for the objective $\text{LimInfAvg} \geq 0$ is in NP.

Proof. By Lemma 83 it is enough to guess a memoryless modular strategy (the memoryless modular strategy is the polynomial witness) and verify that it is indeed a winning strategy. The verification can be achieved in polynomial time using the polynomial-time algorithms of Section 7.1 for WPSs with mean-payoff objectives.

Strict inequalities and stack boundedness. Note that by Corollary 4, the results of Lemma 85 and Theorem 19 also hold for the $\text{LimSupAvg} \geq 0$ objective. For modular strategies we only presented the result for mean-payoff objectives with non-strict inequalities. The results for strict inequalities follow from an adaptation of the proofs for non-strict inequalities (for which we prove memoryless modular strategies are sufficient). Moreover, the results also follow for mean-payoff objectives with the stack boundedness condition for the following reason: we observe that the manipulated operations never increase the stack height. Thus from our results it follow that if there is a modular winning strategy to ensure the mean-payoff objective along with stack boundedness, then there is also a memoryless modular winning strategy. Hence the NP upper bound follows for strict inequalities as well as for stack boundedness.
8.1.3 NP-hardness of the modular winning strategy problem

In this section we establish the NP-hardness of the modular winning strategy problem. Our hardness result will be for one-player WRGs (player-1 WRGs), where every module will have single exits, and the weights are \{-1, 0, +1\}. In other words, our hardness result shows that even a very simple version of the problem (single exit one-player WRGs with constant weights) is NP-hard.

**Reduction.** We present a reduction from the 3-SAT problem (satisfiability of a CNF formula where every clause has exactly three distinct literals). Consider a 3-SAT formula \( \varphi(x_1, x_2, \ldots, x_n) = \bigwedge_{i=1}^{m} c_l_i \) over \( n \) variables \( x_1, x_2, \ldots, x_n \), and \( m \) clauses \( c_l_1, c_l_2, \ldots, c_l_m \). A literal is a variable \( x_i \) or its negation \( \neg x_i \). We construct a player-1 WRG as follows: \( A_\varphi = (A_0, x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n, c_l_1, c_l_2, \ldots, c_l_m) \) in the following way: there is an initial module \( A_0 \), there is a module for every literal, and for every clause. We now describe the modules.

**Module \( A_0 \).** The module invokes in an infinite loop in sequence the modules \( c_l_1, c_l_2, \ldots, c_l_m \), and all the transitions in this module have weight zero.

**Module for clause \( c_l_i \).** There is an edge from the entry node of module \( c_l_i \) to a box that invokes module \( y_i \), for every literal \( y_i \) that appears in the clause \( c_l_i \). There is also an edge from the return node of \( y_i \) to the exit node of \( c_l_i \). All the weights in this module are zero.

**Module for literal \( y_i \).** The entry node of \( y_i \) has outdegree two (left edge and right edge). The left edge is the FALSE edge, which leads to the exit node, and has a weight \(-1\). The right edge is the TRUE edge, which leads to a box that invokes a call for module \( \neg y_i \), and its weight is \(-1\). The return of the box leads to the exit node and the edge weight is \(+2\). The reduction is illustrated pictorially in Figure 8.4.

**Observation 3.** Every path (that is generated by a modular strategy) from the entry
node of the module \( y_i \) to its exit node, has a total weight of at most 0, hence every path (that is generated by a modular strategy) from the entry node of module \( cl_i \) to its exit node, has a total weight of at most 0.

**Lemma 86.** There exists a modular winning strategy for player 1 in \( A_\varphi \) for the objective \( \text{LimInfAvg} \geq 0 \) iff \( \varphi \) is satisfiable.

**Proof.** We first observe that every modular strategy in \( A_\varphi \) is a memoryless modular strategy. Observe that modular strategy for player 1 is a selection of a literal for every clause module, and selecting either the **TRUE** or **FALSE** edge for every literal module. We now present both directions of the proof.

- **Modular winning strategy implies satisfiability.** Let \( \tau \) be a modular winning strategy for player 1, such that the literal \( y^{\tau}_i \) is chosen for clause \( cl_i \). First, towards a contradiction, let us assume that there exists \( i \in \{1, \ldots, m\} \) such that \( \tau \) selects the **FALSE** edge for module \( y_i^{\tau} \). Hence the weight of the path inside the module \( cl_i \) is negative. Thus, due to Observation 3, the mean-payoff value according to \( \tau \) is negative. Therefore \( \tau \) selects the **TRUE** edge for the module \( y_i^{\tau} \). Next, towards a contradiction, let us assume that there exist \( i, j \in \{1, \ldots, m\} \) such that \( y_j^{\tau} = \neg y_i^{\tau} \) and \( \tau \) selects the **TRUE** edge for both modules. In this case, the play will never exit module \( y_i^{\tau} \), and will go forever through edges with negative weights. Therefore if \( \tau \) selects the **TRUE** edge for \( y_i^{\tau} \), then it does not select the **TRUE** edge for \( \neg y_i^{\tau} \). Due to the above, the assignment that assigns a true value to the literal \( y_i^{\tau} \) in clause \( cl_i \) is a valid (non-conflicting) assignment that satisfies \( \varphi \).

- **Satisfiability implies modular winning strategy.** Let \( \overline{\pi} \) be a satisfying assignment (a non-conflicting assignment of truth values to variables) for the formula \( \varphi \). We construct a modular winning strategy \( \tau_{\overline{\pi}} \) as follows. In module \( cl_i \), the modular strategy invokes the module \( y_i^{\overline{\pi}} \), where \( y_i^{\overline{\pi}} \) is a literal for which \( \overline{\pi} \) assigns a true value (since \( \overline{\pi} \) is a satisfying assignment such a literal must exist). In module \( y_i \), follow the **TRUE** edge if \( \overline{\pi} \) assigns a true value to the literal \( y_i \), and follow the **FALSE** edge otherwise. It is straightforward to verify that the mean-payoff of a play according to \( \tau_{\overline{\pi}} \) is zero.

This completes the proof.

Observe that in the hardness reduction we have used positive weight +2 for simplicity, which can be split into two edges of weight +1 each. Hence we have the following theorem.

**Theorem 20.** The modular winning strategy problem is NP-hard for one-player WRGs (player-1 WRGs) with single exit for every module and objective \( \text{LimInfAvg} \geq 0 \) with edge weights in \( \{-1, 0, +1\} \).
Strict inequalities and stack boundedness. We first observe that the above reduction also holds for $\text{LimSupAvg} \geq 0$ objective. Moreover, whenever the 3-SAT formula $\varphi$ is satisfiable, then the witness memoryless modular strategy along with mean-payoff objective also ensures stack boundedness. Hence the hardness result follows for mean-payoff objectives with non-strict inequalities as well as for stack boundedness. The result for strict inequality is obtained as follows: we modify the above reduction by changing the weight of the edge back to the entry node of $A_0$ from 0 to 1. Then if the formula $\varphi$ is satisfiable, then the average payoff for memoryless modular strategies is at least $\frac{1}{|V|}$, where $|V|$ is the number of vertices, and if the formula $\varphi$ is not satisfiable, then the mean-payoff under all memoryless modular strategies is at most 0. Hence the hardness follows also for mean-payoff objectives with strict inequalities. We have the following theorem summarizing the results for modular strategies.

**Theorem 21.** The following assertions hold for WRGs with objectives $\Phi \triangleleft \triangleright 0$, for $\triangleleft \triangleright \in \{\geq, >\}$, $\Phi \in \{\text{LimSupAvg}, \text{LimInfAvg}\}$, as well as objectives $\Phi \triangleleft \triangleright 0$ along with stack boundedness.

1. If there is a modular winning strategy, then there is a memoryless modular winning strategy.

2. The decision problem of whether there is a memoryless modular winning strategy is NP-complete.

3. The decision problem is NP-hard for player-1 WRGs with single exit for every module and edge weights in $\{-1, 0, +1\}$.

### 8.2 Undecidability for (Conjunctive) Multidimensional Mean-Payoff Objectives

In this section we will show that the problem of deciding the existence of modular winning strategy for player 1 in WRGs with multidimensional mean-payoff objectives is undecidable. The reduction would be from reachability games over tuples of integers. We start by introducing these games.

**Reachability games over $\mathbb{Z}^k$.** A reachability game over $\mathbb{Z}^k$ consists of a finite-state game graph $G$, a $k$ dimensional weight function $w : E \to \mathbb{Z}^k$, and an initial weight vector $\vec{\nu} \in \mathbb{Z}^k$. An infinite play $\pi$ is winning for player 1 if there exists some finite prefix $\pi' \sqsubseteq \pi$ such that $w(\pi') + \vec{\nu} = 0$ and the last vertex in $\pi'$ is a player-1 vertex.

**Lemma 87.** The following problem is undecidable: Given a reachability game over $\mathbb{Z}^2$ and a starting vertex $v$, decide if there is a winning strategy $\tau$ for player 1.

**Proof.** We make a simple observation that the undecidability proof for reachability games over $\mathbb{N}^2$ (e.g., see [2]) is easily extended to games over $\mathbb{Z}^2$. 

We will present a general reduction from reachability games over $\mathbb{Z}^k$ to WRGs under modular strategies with multidimensional mean-payoff objectives of $2 \cdot k + 2$ dimensions,
with three modules (two of them with single exit, and an initial module without any exits). Given a reachability game over $\mathbb{Z}^k$ with game graph $G$, weight function $w$ and
initial vector $\vec{\nu}$, we construct a WRG graph $A = \langle A_0, A_1, A_2 \rangle$ with a weight function of $2 \cdot k + 2$ dimensions in the following way.

- **Module $A_0$:** This module repeatedly invokes $A_1$ and $A_2$ (one call to $A_1$ and one call to $A_2$); and all the weights of the transitions are 0.

- **Module $A_1$:** This module has three nodes: entrance, exit and an additional one with a self-loop edge with weight 0 in the first $2 \cdot k$ dimensions, weight +1 in dimension $2 \cdot k + 1$ and weight $-1$ in dimension $2 \cdot k + 2$; the weight of the edges from the entrance node to the additional node and from the additional node to the exit node are 0 in every dimension. All the nodes are in the control of player 1.

- **Module $A_2$:** The nodes of this module are the entrance and exit nodes, the nodes $V$ of the reachability game $G$, and an additional node $v^*$. The entrance node leads to the initial vertex of $G$ with edge weight $(\vec{\nu}, -\vec{\nu}, 0, 0)$ (i.e., the first $k$ dimensions are according to $\vec{\nu}$, dimensions $k + 1$ to $2 \cdot k$ are according to $-\vec{\nu}$, and the last two dimensions are 0). For every edge $e = (u, v)$ in $G$, there is such transition in $A_2$ with weight $(w(e), -w(e), -1, +1)$. In addition, from every player-1 vertex in $V$ there is a transition to $v^*$ with weight 0 in every dimension. In $v^*$ there is a self-loop transition with weight $-1$ in dimension $2k + 1$, $+1$ in dimension $2k + 2$ and 0 in the rest of the dimensions; and in addition there is a transition to the exit node with weight 0 in every dimension.

The pictorial descriptions of the modules $A_0, A_1, A_2$ are shown in Figure 8.5, Figure 8.6, and Figure 8.7, respectively.

**Observation 4.** The following observations hold:

1. If player-1 strategy $\tau_1$ for module $A_1$ is to never exit, then it is not a winning strategy (since the mean-payoff in dimension $2 \cdot k + 2$ will be $-1$.)

2. If for a player-1 strategy $\tau_2$ for module $A_2$, there is a play $\pi$ consistent with $\tau_2$ that does not reach $v^*$, then $\tau_2$ is not a winning strategy (since the mean-payoff of $\rho$ in dimension $2 \cdot k + 1$ will be $-1$.)

**Lemma 88.** If player 1 does not have a winning strategy in the reachability game over $\mathbb{Z}^k$, then there is no modular winning strategy for player 1 in $A$.

**Proof.** If player 1 does not have a winning strategy in the reachability game over $\mathbb{Z}^k$, then let $\sigma$ be a player-2 winning strategy for the reachability game. We fix player-2 strategy for the modular game to be $\sigma$ according to the local history of $A_2$ and claim that it is a winning strategy for player 2 in the WRG against the multidimensional mean-payoff objective for player 1. Indeed, let $\tau = \{\tau_1, \tau_2\}$ be a player-1 modular strategy, and we consider the path $\pi$ which is formed by playing according to $\tau$ and $\sigma$. By Observation 4 if $\pi$ never exit $A_1$ or never reach node $v^*$, then player 2 wins. Otherwise, since $\sigma$ is a winning strategy in the reachability game, we get that in the
first sub-path of \( \pi \) that leads from the entrance of \( A_2 \) to \( v^* \), one of the dimensions \( 1 \leq i \leq 2 \cdot k \) has a negative weight. We note that both \( \sigma \) and \( \tau \) are modular strategies, and thus the path \( \pi \) is periodic and the mean-payoff of \( \pi \) in dimension \( i \) is negative. To conclude, if player 2 is the winner in the reachability game, then player 1 does not have a modular winning strategy in \( A \).

\[ \square \]

**Lemma 89.** If player 1 has a winning strategy in the reachability game, then there is a modular winning strategy for player 1 in \( A \).

**Proof.** Let \( \tau_G \) be a player-1 winning strategy for the reachability game. By König’s Lemma there exists a fixed constant \( n \in \mathbb{N} \) such that player 1 can assure the reachability objective, against every player-2 strategy, with at most \( n \) rounds. We now derive a modular winning strategy in \( A \) from \( \tau_G \):

- **Module \( A_1 \):** Follow the self-loop edge for \( n \) rounds and exit.
- **Module \( A_2 \):** Follow strategy \( \tau_G \), until the weight in every dimension, according to the reachability game over \( G \), is 0 and a player-1 vertex was reached, and then go to \( v^* \). Let \( m \) be the number of rounds played according to \( \tau_G \) in the current local history of \( A_2 \), then player 1 follows the self-loop in \( v^* \) for \( n - m \) times and goes to the exit node.

It is easy to observe that any play according to the strategy above has a mean-payoff value of 0 in every dimension. \( \square \)

From Lemma 87, Lemma 88, and Lemma 89 we obtain the following result:

**Theorem 22.** The problem of deciding the existence of a modular winning strategy in WRGs with multidimensional mean-payoff objectives is undecidable, even for hierarchical games (i.e., games without recursive calls), with six dimensions, three modules and with at most single exit for each module.
Part III

Applications
Chapter 9

The Complexity of Infinitely Repeated Alternating Move Games

In this chapter, we investigate infinitely repeated games in which players alternate making moves. This framework can model, for example, five telecommunication providers competing for customers: each company can observe the price that is set by the others, and it can update the price at any time. In the short term, each company can benefit from undercutting its opponents price, but since the game is repeated indefinitely, in some settings, it might be better to coordinate prices with the other companies. Such examples motivate us to study equilibria in alternating move games.

In this work, we study infinitely repeated $k$-player $n$-action games. In such games, in every round, a player chooses an action, and the utility of each player (for the current round) is determined according to the $k$-tuple of actions of the players. Each player goal is to maximize his own long-run average utility as the number of rounds tends to infinity.

These games were studied by Roth et al. in [85], and they showed an FPTAS (fully polynomial-time approximation scheme, which requires the algorithm to be polynomial in both the problem size and $\frac{1}{\epsilon}$) for computing an $\epsilon$-equilibrium. Their result provided a theoretical separation between the alternating move model and the simultaneous move model, since for the latter, it is known that an FPTAS for computing approximate equilibria does not exists for games with $k \geq 3$ players unless $P=PPAD$. Their result was obtained by a simple reduction to mean-payoff games on graphs. These games were presented in [47], and they play an important rule in automata theory and in economics. The computational complexity of finding an exact equilibrium for such games is a long standing open problem, and despite many efforts [14, 26, 54, 69, 78, 110], there is no known polynomial solution for this problem.

We extend the work of [85] by investigating the complexity of an exact equilibrium (which was stated as an open question in [85]), and by investigating the computational complexity of finding an $\delta$-optimal approximated equilibrium with respect to the social welfare metric. Our main technical results are as follows:

- We show a reduction from mean-payoff games on graphs to two-player zero-sum alternating move games, and thus we prove that $k$-player alternating move games
• We show that optimal equilibrium can be obtained by pure strategies, and we show an FPTAS for computing an $\delta$-optimal $\epsilon$-equilibrium. In addition, we show that computing an exact optimal equilibrium is polynomial time equivalent to solving mean-payoff games on graphs.

We note that the first result may suggest that two-player alternating move games are harder than two-player simultaneous move games, since a polynomial time algorithm to solve the latter is known [71]. Hence, along with the result of [85], we get that simultaneous move games are easier to solve with comparison to alternating move games for two-player games, and are harder to solve for $k \geq 3$ player games.

This chapter is organized as following. In the next section we bring formal definitions for alternating move games and recall the definitions of mean-payoff games on graphs. In Section 9.2, we show that alternating move games are at least as hard as mean-payoff games on graphs. In Section 9.3 we investigate the properties of optimal equilibria, and we present an FPTAS for computing an $\delta$-optimal $\epsilon$-equilibrium.

**Bibliographic note.** The results of this chapter were first published in: Yaron Velner: The Complexity of Infinitely Repeated Alternating Move Games. ICALP 2013.

### 9.1 Definitions

In this section we bring the formal definitions for alternating move repeated games and mean-payoff games on graphs. Alternating move games are presented in Subsection 9.1.1, and mean-payoff games are presented in Subsection 9.1.2.

#### 9.1.1 Alternating move repeated games

**Actions, plays and utility function.** A $k$-player $n$-action game is defined by an action set $A_i$ for every player $i$, and by $k$ utility functions, one for each player, $u_i : A_1 \times \ldots \times A_k \to [-1, 1]$. W.l.o.g we assume that the size of all action set is the same, and we denote it by $n$. We note that any game can be rescaled so its utilities are bounded in $[-1, 1]$, however, the FPTAS that was presented in [85], and our results in Subsection 9.3.4 crucially rely on the assumption that the utilities are in the interval $[-1, 1]$.

An alternating move game is played for infinitely many rounds. In round $t$ player $j = 1 + (t \mod k)$ plays action $a_t^j$, and a vector of actions $a^t = (a_1, \ldots, a_k)$ is produced, where $a_i \in A_i$ is the last action of player $i$. In every round $t$, player $i$ receives a utility $u_i(a^t)$, which depends only on the last action of each of the $k$ players (W.l.o.g the utility in the first $k$ rounds is zero for all players). An infinite sequence of rounds forms a **play**, and we characterize a play either by an infinite sequence of actions or by the corresponding sequence of vectors of actions. The utility of player $i$ in a play $a^1 a^2 \ldots a^t a^{t+1} \ldots$ is the **limit average payoff**, namely, $\lim_{n \to \infty} \frac{1}{n} \sum_{t=1}^n u_i(a^t)$. When this limit does not exist, we define the utility of the play for player $i$ to be $\liminf_{n \to \infty} \frac{1}{n} \sum_{t=1}^n u_i(a^t)$. We note that in the frame work of [85], the utility of a play was undefined when the limit does
not exist. The results we present in this chapter for the lim inf metric holds also for the framework of \[^{[85]}\]. On the other hand, if we would take the lim sup value instead, then the problem is much easier. An optimal equilibrium is obtained when all players join forces and maximize player 1 + (i (mod k)) utility for 2^i rounds (for i = 1, 2, ..., ∞). Hence, we can easily produce a polynomial algorithm to solve lim sup games.

Strategies. A strategy is a recipe for player’s next action, based on the entire history of previous actions. Formally, a (mixed) strategy for player i is a function \(\sigma_i : (A_1 \times \cdots \times A_k)^* \times A_1 \times \cdots \times A_{i-1} \rightarrow \Delta(A_i)\), where \(\Delta(S)\) denotes the set probability distribution over any finite set S. We say that \(\sigma_i\) is a pure strategy if \(\Delta\) is a degenerated distribution. A strategy profile is a vector \(\sigma = (\sigma_1, \ldots, \sigma_k)\) that defines a strategy for every player. A profile of pure strategies uniquely determines the action vector in every round and yields a utility vector for the players. A profile of mixed strategies determines, for every round \(t\) in the play, a distribution of sequences of action vectors, and the average payoff in round \(t\) is the expected average payoff over the distribution of action vectors. Formally, for a strategy profile \(\sigma\) we denote the average payoff of player i in round \(t\) by

\[
P_{i,t}(\sigma) = \frac{\sum_{a^1, a^2, \ldots, a^t \sim \sigma} u_i(a^1) + \cdots + u_i(a^t)}{t}
\]

and the utility of player i is lim inf_{t→∞} \(P_{i,t}\).

Equilibria, ε equilibria and optimal equilibria A strategy profile forms an equilibrium if none of the players can strictly improve his utility (that is induced by the profile) by unilaterally deviating from his strategy (that is defined by the profile). For every \(\epsilon > 0\), we say that a strategy profile forms an \(\epsilon\)-equilibrium if none of the players can improve his utility by more than \(\epsilon\) by unilaterally deviating from his strategy.

The social welfare of a strategy profile is the sum of the utilities of all players. An equilibrium (resp. an \(\epsilon\)-equilibrium) is called an optimal equilibrium (optimal \(\epsilon\)-equilibrium) if its social welfare is not smaller than the social welfare of any other equilibrium (\(\epsilon\)-equilibrium). For \(\delta > 0\), an equilibrium (resp. an \(\epsilon\)-equilibrium) is called an \(\delta\)-optimal equilibrium if its social welfare is not smaller by more than \(\delta\) with comparison to the social welfare of any other equilibrium (\(\epsilon\)-equilibrium).

9.1.2 Mean-payoff games on graphs

Plays and payoffs. A mean-payoff game on a graph is defined by a weighted directed bipartite graph \(G = (V = V_1 \cup V_2, E, w : E \rightarrow \mathbb{Q})\) and an initial vertex \(v_0 \in V\). The game consists of two players, namely, maximizer (who owns \(V_1\)) and minimizer (who owns \(V_2\)). Initially, a pebble is place on the initial vertex, and in every round, the player who owns the vertex in which the pebble resides, advance the pebble into an adjacent vertex. This process is repeated forever and forms a play. A play is characterized by a sequence of edges, and the average payoff of a play \(\rho = e_1 \ldots e_t\) up to round \(t\) is denoted by \(P_t = \frac{1}{t} \sum_{i=1}^{t} w(e_i)\). The value of a play is the limit average payoff (mean-payoff), namely, lim inf_{t→∞} \(P_t\). (We note that for games on graphs, the lim sup metric
gives the same complexity results.) The objective of the maximizer is to maximize the mean-payoff of a play, and the minimizer aims to minimize the mean-payoff.

**Strategies, memoryless strategies, optimal strategies and winning strategies.** In this work, we consider only pure strategies for games on graphs, and it is well-known that randomization does not give better strategies for mean-payoff games. A *strategy* for maximizer is a function $\sigma : (V_1 \times V_2)^* \times V_1 \to E$ that decides the next move, and similarly, for the minimizer a strategy is a function $\tau : (V_1 \times V_2)^* \to E$. A strategy is called *memoryless* if it depends only on the current position of the pebble. Formally, a memoryless strategy for the maximizer is a function $\sigma : V_1 \to E$ and similarly a memoryless strategy for the minimizer is a function $\tau : V_2 \to E$.

A profile of strategies $(\sigma, \tau)$ uniquely determines the mean-payoff value of a game. We say that a play $\pi = e_1 e_2 \ldots e_n \ldots$ is *consistent* with a maximizer strategy $\sigma$ if there exists a minimizer strategy $\tau$ such that $\pi$ is formed by $(\sigma, \tau)$. We say that the *value* of a maximizer strategy is $p$ if it can assure a value of at least $p$ against any minimizer strategy. Analogously, we say that the value of a minimizer strategy is $p$ if it can assure a value of at most $p$ against any maximizer strategy.

We say that a maximizer strategy is *optimal* if its value is maximal (with respect to all possible maximizer strategies). Analogously, a minimizer strategy is optimal if its value is minimal. For a given threshold, we say that a maximizer strategy is a *winning* strategy if it assures mean-payoff value that is greater or equal to the given threshold, and a minimizer strategy is winning if it assures value that is strictly smaller than the given threshold.

**One-player games, and games according to memoryless strategies** A special (and easier) case of games on graphs is when the out-degree is one for all the vertices that are owned by a certain player. In this case, all the choices are done by one player. For a two-player game on graph $G$, and a player-1 memoryless strategy $\sigma$, we define the one-player game graph $G^\sigma$ to be the game graph that is formed by removing, for every player-1 vertex $v$, the out-edges that are not equal to $\sigma(v)$.

**Classical results on mean-payoff games.** Mean-payoff games were introduced in ’79 by Ehrenfeucht and Mycielski [47], and their main result was that optimal strategies (for both players) exist, and moreover, the optimal value can be obtained by a memoryless strategy. The decision problem for mean-payoff games is to determine if the maximizer has a winning strategy with respect to a given threshold. The existence of optimal memoryless strategies almost immediately proves that the decision problem for mean-payoff games is in $\text{NP} \cap \text{coNP}$, and thus it is unlikely to be $\text{NP}$-hard (or $\text{coNP}$-hard). Zwick and Paterson [110] introduced the first pseudo-polynomial algorithm, which runs in polynomial time when the weights of the edges are encoded in unary. They also provided a polynomial algorithm for the special case of one-player mean-payoff games. A randomized sub-exponential algorithm for mean-payoff games is also known [14], but despite many efforts, the existence of a polynomial algorithm to
solve mean-payoff games remains an open question, and it is one of the rare problems in computer science that is known to be in $\text{NP} \cap \text{coNP}$ but no polynomial algorithm is known.

We summarize the known results on mean-payoff games in the next theorems. The first theorem states that optimal strategies exist and moreover, there exist optimal strategies that are memoryless.

**Theorem 23 ([47]).** For every mean-payoff game there exists a maximizer memoryless strategy $\sigma$ and a minimizer memoryless strategy $\tau$ such that $\sigma$ is optimal for the maximizer and $\tau$ is optimal for the minimizer.

The next theorem shows that there is a polynomial algorithm that computes optimal strategies if and only if there is a polynomial algorithm for the mean-payoff games decision problem.

**Theorem 24 ([110]).** The following problems are polynomial time inter-reducible: (i) Compute maximizer optimal memoryless strategy. (ii) Compute the optimal value that maximizer can assure. (iii) Determine whether maximizer optimal value is at least zero. (iv) Determine whether maximizer optimal value is greater than zero.

### 9.2 Two-Player Zero-Sum (Alternating Move) Games are Inter-Reducible with Mean-Payoff Games

In this section we prove that there is a polynomial algorithm that computes an exact equilibrium for two-player zero-sum (alternating move) games if and only if there exists a polynomial algorithm that solves mean-payoff games.

The reduction from two-player zero-sum games to mean-payoff games is trivial, for a two-player zero-sum game with actions $A_1, A_2$ and utility functions $u_1 : A_1 \times A_2 \to \mathbb{Q}$ and $u_2 = -u_1$, we construct a complete bipartite game graph $G = (V = A_1 \cup A_2, E = (A_1 \times A_2) \cup (A_2 \times A_1), w : E \to \mathbb{Q})$ such that the weight of the transition from $a_1 \in A_1$ to $a_2 \in A_2$ is simply $u_1(a_1, a_2)$, and the weight of the transition from $a_2$ to $a_1$ is also $u_1(a_1, a_2)$. It is a simple observation that a pair of optimal strategies (for the maximizer and minimizer) in the mean-payoff game induces an equilibrium strategy profile in the two-player zero-sum game and vice versa.

The reduction for the converse direction is more complicated. For this purpose we bring the notion of undirected game graph. A mean-payoff game graph is said to be undirected if its edge relation is symmetric, and $w(v_1, v_2) = w(v_2, v_1)$ for every edge $(v_1, v_2)$. (Basically, it is a game on an undirected graph.) The next simple lemma shows a reduction from mean-payoff games on a complete bipartite undirected graphs to two-player zero-sum game.

**Lemma 90.** There is a polynomial reduction from mean-payoff games on complete bipartite undirected graphs to two-player zero-sum games.

**Proof.** The proof is straightforward. Let $V_1$ and $V_2$ be the maximizer and minimizer (resp.) vertices in the mean-payoff game. We construct a two-player zero-sum game in
the following way. The set of action of player 1 is $A_1 = V_2$ and the set of action for player 2 is $A_2 = V_1$. We denote by $W$ the least value for which all the weights in the undirected graphs are in $[-W, +W]$, and the utility function of player 1 is $u_1(a_1, a_2) = \frac{w(a_1, a_2)}{W} = \frac{w(a_2, a_1)}{W}$, and $u_2 = -u_1$. It is trivial to observe that an equilibrium profile induces a pair of optimal strategies for the mean-payoff game, and the proof follows.

Due to Lemma 90, all that is left is to prove that mean-payoff games on complete bipartite undirected graphs are equivalent to mean-payoff games. A recent result by Chatterjee, Henzinger, Krinninger and Nanongkai [36] gives us the first step towards such proof.

**Theorem 25** (Corollary 24 in [36]). Solving mean-payoff games on complete bipartite (directed) graphs is as hard as solving mean-payoff games on arbitrary graphs.

We use the above result as a black box and extend it to complete bipartite undirected graphs. We note that the main difference between directed and undirected graphs is that for undirected graphs the weight function is symmetric. In the rest of this section we will describe a process that for a given complete bipartite directed graph, generates a suitable symmetric weight function, and the winner in the generated graph is the same as in the original graph.

We say that a directed game graph has a normalized weight function if it assigns a positive weight to every out-edge of maximizer vertex, and a negative weight for every out-edge of minimizer vertex. The next lemma shows that we may assume w.l.o.g that a directed game graph has a normalized weight function.

**Lemma 91.** Solving mean-payoff games on (directed) bipartite graphs is polynomial time inter-reducible to solving mean-payoff games on (directed) bipartite graphs with normalized weights.

**Proof.** Let $G$ be a non-normalized graph and let us denote by $W$ the heaviest weight (in absolute value) that is assigned by its weight function $w$. We construct a normalized graph $G'$ from $G$ by defining a weight function $w'$ as:

$$w'(u, v) = \begin{cases} w(u, v) + (W + 1) & \text{if } u \text{ is owned by maximizer} \\ w(u, v) - (W + 1) & \text{otherwise (if } u \text{ is owned by the minimizer)} \end{cases}$$

Clearly, $G'$ is a normalized graph, and since $G'$ and $G$ are bipartite, it is straight forward to observe that for any finite path in $\pi$ we have that $|w(\pi) - w'(\pi)| \leq W + 1$, and thus, for every infinite path $\rho$, we have that the mean-payoff value of $\rho$ according to $w$ is identical to the mean-payoff of $\rho$ according to $w'$. Therefore, a maximizer winning (resp. optimal) strategy in graph $G$ is a winning (optimal) strategy also in $G'$ and vice versa, and the proof of the lemma follows.

$\square$
In the next lemma we show that mean-payoff games on direct normalized bipartite complete graphs are as hard as mean-payoff games on undirected normalized bipartite complete graphs.

**Lemma 92.** The problem of determining whether maximizer has a winning strategy for a threshold 0 for a mean-payoff games on a directed normalized bipartite complete graph is as hard as the corresponding problem for mean-payoff games on an undirected normalized bipartite complete graph.

**Proof.** We show that for a given directed normalized bipartite complete graph $G = (V,E,w)$ we can construct (in polynomial time) an undirected normalized bipartite complete graph $G' = (V',E',w')$ such that maximizer has a winning strategy in $G$ (for threshold 0) if and only if he has a winning strategy in $G'$ (again for threshold 0).

Informally, we build $G'$ from $G$ by taking the same set of vertices and by assigning a weight $w'(u,v)$ that matches either to $w(u,v)$ or to $w(v,u)$.

To formally construct $w'$ we present the notion of *surely loosing edges*. Recall that $V_1$ are maximizer vertices and $V_2$ are minimizer vertices. We say that a maximizer vertex out-edge $(v_1,v_2)$ is *surely loosing for maximizer* if $w(v_1,v_2) + w(v_2,v_1) < 0$. We claim that if maximizer has a winning strategy, then all its memoryless winning strategies do not contain surely loosing edges. That is, for every memoryless winning strategy $\sigma$ and a surely loosing edge $(v_1,v_2)$ we have $\sigma(v_1) \neq (v_1,v_2)$. Indeed, towards contradiction let us assume that $\sigma(v_1) = (v_1,v_2)$, and recall that $G$ is a complete bipartite graph, than for the minimizer strategy $\tau$ that leads from every minimizer vertex to $v_1$ we have that the mean-payoff value of $(\sigma,\tau)$ is $\frac{w(v_1,v_2) + w(v_2,v_1)}{2} < 0$, in contradiction to the assumption that $\sigma$ is a winning strategy. Analogously, $(v_2,v_1)$ is a surely loosing edge for minimizer if $w(v_1,v_2) + w(v_2,v_1) \geq 0$ and by the same arguments there is no memoryless winning strategy for minimizer that contains a surely loosing edge.

We are now ready to formally define $G' = (V',E',w')$. We define $V' = V$, and since $G'$ is complete bipartite the definition of $E'$ follows immediately. For an edge $\{v_1,v_2\} \in E'$, where $v_i \in V_i$, we define

$$w'(v_1,v_2) = \begin{cases} w(v_1,v_2) & \text{if } w(v_1,v_2) + w(v_2,v_1) \geq 0 \\ w(v_2,v_1) & \text{otherwise (if } w(v_1,v_2) + w(v_2,v_1) < 0) \end{cases}$$

We first prove that if $\sigma$ is a memoryless winning strategy for maximizer in $G$, then it is also a winning strategy in $G'$. Indeed, let $\rho$ be a path (in $G$ and in $G'$) that is consistent with $\sigma$. We claim that for every path $\pi$ that is a finite prefix of $\rho$ we have $w(\pi) \leq w'(\pi)$ (that is, its weight in $G'$ is not less than its weight in $G$), and we prove it by a simple induction on the length of $\pi$. For $|\pi| = 0$ the claim is trivial. For $|\pi| > 0$ let $(u,v)$ be the last edge in $\pi$. If $u \in V_1$, then we get that $\sigma(u) = (u,v)$ and therefore $(u,v)$ is not a surely loosing edge. Therefore, by definition, $w(u,v) = w'(u,v)$ and the claim follows by the induction hypothesis (since it holds for the prefix of length $|\pi| - 1$). If $u \in V_2$, then since $G$ is normalized we have that $w(u,v) < 0$ and $w(v,u) > 0$. Therefore,
by definition, \( w'(u,v) \geq w(u,v) \), and by the induction hypothesis the claim follows. Since \( \sigma \) is a winning strategy we get that the mean-payoff value of \( \rho \) is non-negative according to \( w \), and by the last claim we get that the mean-payoff of \( \rho \) according to \( w' \) is greater or equal to the mean-payoff value according to \( w \). Hence \( \sigma \) is a winning strategy for maximizer also in \( G' \).

By the same arguments we get that if minimizer has a memoryless winning strategy in \( G \), then the same strategy is winning for minimizer also in \( G' \).

Finally, due to Theorem 23, we get that maximizer has a winning strategy in \( G \) if and only if he has a memoryless winning strategy in \( G \) if and only if he has a memoryless winning strategy in \( G' \) if and only if it has a winning strategy in \( G' \) and the proof of the lemma follows.

To conclude, by Lemma 90 we get that a polynomial algorithm for alternating move two-player zero-sum games exists if and only if there exists a polynomial algorithm for solving mean-payoff games on a complete bipartite undirected graph, and by Lemmas 91, 90 and 92 and by Theorem 25 we get that the latter exists if and only if there exists a polynomial algorithm for solving mean-payoff games on arbitrary (directed) graphs. Hence, the main result of this section follows.

**Theorem 26.** There exists a polynomial time algorithm for computing exact equilibrium for two-player zero-sum (alternating move) games if and only if there exists a polynomial time algorithm for solving mean-payoff games on graphs.

### 9.3 Complexity of Computing Optimal Equilibrium

In this section, we investigate the complexity of computing an optimal equilibrium. Our main results are summarized in the next theorem:

**Theorem 27.**

1. Optimal equilibrium can be obtained by a profile of pure strategies.

2. If mean-payoff games are in \( P \), then there is a polynomial algorithm for computing an exact optimal equilibrium.

3. If mean-payoff games are not in \( P \), then there is no FPTAS that approximate the social welfare of the optimal equilibrium.

4. There is an FPTAS to compute an \( \epsilon \)-equilibrium that is \( \delta \)-optimal. (Note that it does not necessarily approximate the value of an exact optimal equilibrium.)

We will prove Theorem 27 in the next four subsections: In Subsection 9.3.1 we show the naive algorithm for computing an equilibrium that is based on Folk Theorem, and we prove basic properties of equilibria in alternating move games. In Subsection 9.3.2 we prove Theorem 27(1). In Subsection 9.3.3 we investigate the complexity of computing the social welfare of the optimal equilibrium, and prove Theorem 27(2) and
Theorem 27(3). Finally, in Subsection 9.3.4 we prove Theorem 27(4) which is the main result of this section.

In this section, we will model n-action k-player alternating move games by a multi-weighted graph, according to the following conventions: The vertices of the graph are the vertices in the set \( V = (A_1 \times A_2 \times \cdots \times A_k) \times \{1, \ldots, k\} \), and we say that player \( i \) owns the vertex set \( V_i = (A_1 \times A_2 \times \cdots \times A_k) \times \{i\} \). Intuitively, a vertex is characterized by an action vector and by a player that owns it. The pair \((u, v)\) is in the edge relation if \( u \) is owned by player \( i \), \( v \) is owned by player \( i+1 \) (where player \( k+1 \) is player 1), and there is at most one difference in the action vector of \( u \) and \( v \) and it is in position \( i \). The weight of every edge is a vector of size \( k \) that corresponds to the utility vector of the actions.

Formally, if \( u = (\vec{a}_1, i) \) and \( v = (\vec{a}_2, i + 1) \) then \( w(u, v) = (u_1(\vec{a}_2), u_2(\vec{a}_2), \ldots, u_k(\vec{a}_2)) \).

For an infinite path in the multi-weighted graph we define the dimension \( i \) of mean-payoff vector of the path to be the mean-payoff value of the path according to dimension \( i \). It is an easy observation that every infinite path in the graph corresponds to a play and its mean-payoff vector corresponds to the utility vector of the play. We note that the size of the graph is \( k^2 \cdot n^k \) which is polynomial in the size of the encoding of the utility functions (which is \( k \cdot n^k \)), hence this graph can be constructed in polynomial time.

### 9.3.1 Basic properties of equilibria

The Folk Theorem gives a conceptually simple (but inefficient) technique to construct an equilibrium. Intuitively, an equilibrium is obtained when each of the players play as if the goal of all the other players is to minimize its utility, and if one of the players deviates from this strategy, then all the other players switch to playing according to a strategy that will minimize the utility of the rebellious player. Formally, let \( G \) be the corresponding k-player game graph that models the alternating move game. For every player \( i \), we consider a zero-sum two-player mean-payoff game graph \( G^i \) in which the maximizer owns player \( i \) vertices and the minimizer owns the other vertices. Let \( \sigma_i \) be an arbitrary optimal strategy for the maximizer in graph \( G^i \), let \( \overline{\sigma}_i \) be an arbitrary optimal strategy for the minimizer in \( G^i \), and let \( \nu_i \) be the value that is obtained by the strategy profile \((\sigma_i, \overline{\sigma}_i)\). Then if every player \( i \) plays according to the strategy:

\[
\begin{align*}
\text{If player } j \neq i \text{ deviated from } \sigma_j, \text{ then play according to } \overline{\sigma}_j \text{ forever, and} \\
\text{otherwise play according to } \sigma_i
\end{align*}
\]

an equilibrium is formed (since by definition, playing according to \( \sigma_i \) assures utility at least \( \nu_i \), and deviating from \( \sigma_i \) assures utility at most \( \nu_i \)).

In the next lemma, we extend the basic principle of Folk Theorem and get a characterization of all the equilibria that are obtained by a profile of pure strategies.

**Lemma 93.** Let \((t_1, t_2, \ldots, t_k)\) be a utility vector such that \( t_i \geq \nu_i \) (for every player \( i \)), then there exists a pure equilibrium with utility exactly \( t_i \) for every player \( i \) if and only if there exists an infinite path \( \pi \) in the graph \( G \) with mean-payoff vector \((t_1, t_2, \ldots, t_k)\).
Proof. The direction from left to right is trivial, since a profile of pure strategies has a (unique) infinite path in the graph that is consistent with the strategies. To prove the converse direction we introduce the notion of a path strategy and the notion of path equilibrium. For a path \( \pi \) we define the path strategy for player \( i \) to be: at round \( j \) (which is a player \( i \) round), play according to the \( j \)-th edge in \( \pi \). And we define the strategy \( \sigma_{\pi}^i \) to be: If player \( j \neq i \) deviated from the path strategy of \( \pi \), then play forever according to \( \sigma_j \), and otherwise play according to the path strategy of \( \pi \). The path equilibrium of the infinite path \( \pi \) is the profile \( (\sigma_{\pi}^1, \ldots, \sigma_{\pi}^k) \). The profile is an equilibrium since it assures a utility \( t_i \geq \nu_i \) for every player, and if player \( i \) deviates from the strategy \( \sigma_{\pi}^i \) he will end up with a utility \( \nu_i \).

9.3.2 Optimal equilibrium can be obtained by pure strategies

In this subsection, we extend Lemma 93 also for the case of mixed strategies, and as a consequence we get that optimal equilibrium can be obtained by pure strategies. Intuitively, we wish to show that if a profile of (mixed) strategies yields a utility vector \( (t_1, \ldots, t_k) \), then there exists an infinite path in the graph with mean-payoff vector that is greater or equal (in every dimension) to \( (t_1, \ldots, t_k) \). Then we get that if a utility vector is obtained by a profile of mixed strategies, and then by Lemma 93 it is also obtained by a profile of pure strategies.

We formally prove the above by the next two lemmas.

**Lemma 94.** Let \( G \) be a multi-weighted graph that is strongly connected, and let \( (t_1, \ldots, t_k) \) be a vector. Then if for every \( \alpha > 0 \) there exists a (finite) cyclic path with average weight at least \( t_i - \alpha \) in every dimension, then there exists an infinite path with mean-payoff vector at least \( (t_1, \ldots, t_k) \).

**Proof.** We assume that for every \( \alpha > 0 \) there exists a finite cyclic path \( C_\alpha \) with average weight at least \( t_i - \alpha \) in every dimension, and we let \( v_\alpha \) be an arbitrary vertex in the cycle \( C_\alpha \). For every two vertices \( u \) and \( v \), we denote by \( \pi_{u,v} \) the shortest path from \( u \) to \( v \) (recall that \( G \) is strongly connected), and we denote by \( W \) the size of the biggest weight (in absolute value) in \( G \). Intuitively, we obtain an infinite path with mean-payoff vector at least \( (t_1, \ldots, t_n) \) by following the cycle \( C_\alpha \) for \( \alpha = 1 \), and then we follow the path \( \pi_{v_\alpha,v_{\alpha/2}} \) and we follow the cycle \( C_{\alpha/2} \) twice, and then we follow \( C_{\alpha/4} \) three times and so on. However, we have to make sure that the average payoff does not decrease too much when following a cycle for the first time.

Formally, for every \( i \in \mathbb{N} \), we denote by \( L_i \) the length of the cycle \( C_{\frac{\alpha}{2^i}} \), and by \( m_i = iWL_{i+1} \). We define \( \rho_0 \) to be the empty path, and for every \( i > 0 \) we define \( \rho_i = \rho_{i-1} \pi_{v_i,v_{i+1}} C_{\frac{\alpha}{2^i}}^{m_i} \), and we define \( \rho \) to be the infinite path that is the limit of the sequence \( \rho_0, \rho_1, \ldots, \rho_i, \ldots \). Due to the fact that the length of \( \pi_{v_i,v_{i+1}} \) is bounded (by the size of the graph), and since the maximal weight is at most \( W \) (and the minimal weight is at least \( -W \)), we get, by a simple algebra, that the mean-payoff vector of \( \rho \) is at least \( (t_1, \ldots, t_k) \)
** Lemma 95.** Let $\sigma$ be a profile of (mixed) strategies with utility vector $(t_1, \ldots, t_k)$. Then for every $\alpha > 0$ there exists a cyclic path in the game graph with average weight at least $t_i - \alpha$ in every dimension.

**Proof.** By definition of the utility function, for every $\delta > 0$, there exists a round $j$ such that the expected average utility, in every round after $j$, is at least $t_i - \delta$ in every dimension. We denote by $\Pi_j$ the (finite) set of paths with length $j$ that have non-zero probability according to the strategy profile $\sigma$, and w.l.o.g we assume that all the paths have the same probability (and a path may occur more than once in $\Pi_j$).

For a path $\pi \in \Pi_j$, we denote by $C_\pi$ the longest cyclic path that is a sub-path of $\pi$. We note that since $G$ is a finite graph, we get that $|C_\pi| \geq |\pi| - 2 |G|$ (where $|G|$ denotes the number of vertices in $G$), and in every dimension: $w(C_\pi) \geq w(\pi) - 2 |G|W$ and $\text{Avg}(C_\pi) \geq \text{Avg}(\pi) - \frac{2 |G| W}{|\pi| - 2 |G|}$.

We partition $\Pi_j$ to $|G|$ sets (some of them may be empty) namely $\Pi^v_j$ for every $v \in G$, such that for every path $\pi \in \Pi^v_j$ we have that $v$ is in the cyclic path $C_\pi$. For a set $\Pi^v_j = \{\pi_1, \ldots, \pi_m\}$, we denote $C(\Pi^v_j) = C_{\pi_1}C_{\pi_2} \ldots C_{\pi_m}$ (note that $C(\Pi^v_j)$ is a path). For every two vertices $u, v \in G$, we denote by $\pi_{u,v}$ the shortest path between $u$ and $v$, and we note that $|\pi_{u,v}| \leq |G|$ and $w(\pi_{u,v}) \geq -|G|W$ in every dimension. Finally, we assume that the vertex set of $G$ is $V = \{v_1, \ldots, v_m\}$ and we define the cyclic path

$$\pi = C(\Pi_{v_1}^1)_{v_1,v_2}C(\Pi_{v_2}^2)_{v_2,v_3} \ldots C_{v_{m-1},v_m}C(\Pi_{v_m}^m)_{v_m,v_1}$$

The average weight of $\pi$ in every dimension is at least

$$\text{Avg}(\Pi_j) - \frac{3 |G|^2 W}{j}$$

and since in every dimension $\text{Avg}(\Pi_j) \geq t_i - \delta$, then for $j \geq \frac{3 |G|^2 W}{\delta}$ we get that in every dimension

$$\text{Avg}(\pi) \geq t_i - 2 \delta$$

and for $\delta = \frac{\alpha}{2}$ we get that $\pi$ is a cyclic path with average weight at least $t_i - \alpha$ in every dimension, and the proof of the lemma follows.

We are now ready to prove that the utility vector of a mixed equilibrium can be obtained by a pure equilibrium.

**Proposition 6.** Let $\sigma$ be a profile of mixed strategies that induces a utility vector $(t_1, \ldots, t_k)$. Then there exists a profile $\sigma'$ of pure strategies that induces exactly the same utility vector. Moreover, if $\sigma$ is an equilibrium, then so is $\sigma'$.

**Proof.** By Lemma 95 we get that for every $\alpha > 0$ there is a cyclic path with average weight at least $t_i - \alpha$ in every dimension. Therefore, by Lemma 94 we get that there is
an infinite path in $G$ with mean-payoff vector at least $(t_1, \ldots, t_k)$, and by Lemma 93 we get that there is a profile of pure strategies that has utility at least $(t_1, \ldots, t_k)$. If $\sigma$ is an equilibrium we get that $t_i \geq \nu_i$ (since otherwise, player $i$ would deviate to strategy $\sigma_i$), and thus, by Lemma 93, we get that there is a pure equilibrium that gives the same utility vector.

The next corollary immediately follows from Proposition 6.

**Corollary 5** (Theorem 27(1)). An optimal equilibrium can be obtained by a profile of pure strategies.

### 9.3.3 The complexity of computing the social welfare of the optimal (exact) equilibrium

In this section we show that if there is a polynomial algorithm for mean-payoff games, then there is a polynomial algorithm to compute an optimal equilibrium in a $k$-player alternating move games. We also prove the converse direction, that is, we show that if there is a polynomial algorithm that computes the social welfare of the optimal equilibrium, then there is a polynomial algorithm that solves mean-payoff games. We prove these two assertions in the next two lemmas.

**Lemma 96.** Suppose that mean-payoff games are in P, then there is a polynomial algorithm that computes the social welfare of an optimal equilibrium.

**Proof.** Due to Corollary 5, it is enough to consider only pure strategies, and due to Lemma 93 a vector of utilities $(t_1, \ldots, t_k)$ is obtained by a pure equilibrium if and only if $t_i \geq \nu_i$ (for $i = 1, \ldots, k$) and there is an infinite path in the game graph with mean-payoff $(t_1, \ldots, t_k)$. Since we assume that there is a polynomial algorithm for computing $\nu_i$, our problem boils down to

Find the maximal value of $\sum_{i=1}^k t_i$ subject to

- $t_i \geq \nu_i$; and
- there exists an infinite path with mean-payoff vector at least $(t_1, \ldots, t_k)$

It was shown in [94] (in the proof of Theorem 18) that the problem of deciding whether there exists an infinite path with mean-payoff vector at least $(t_1, \ldots, t_k)$ can be reduced (in polynomial time) to a set of linear constraints. Moreover, the generated set of constraints remain linear even when $t_i$ is a variable. Hence, we can find a feasible threshold vector $(t_1, \ldots, t_k)$ (that is, a vector that is realizable by an infinite path in the graph) that maximizes $\sum_{i=1}^k t_i$ by linear programming. Therefore, if we have a polynomial algorithm that computes $\nu_i$, then we can find the social welfare of the optimal equilibrium in polynomial time.
Lemma 96 proves Theorem 27(2) and gives an upper bound to the complexity of computing optimal equilibrium. In the next lemma we show that this bound is tight, and that the social welfare of the optimal equilibrium cannot be approximated, unless mean-payoff games are in P.

**Lemma 97 (Theorem 27(3)).** There is no FPTAS that approximates the social welfare of an optimal equilibrium, unless mean-payoff games are in P.

**Proof.** Due to Theorem 24 and Theorem 26 it is enough to show that if we could approximate the social welfare of the optimal equilibrium in a three-player game, then we would be able to determine whether in a two-player zero-sum game, player 1 has a strategy that assures a value that is strictly greater than 0. The proof is straightforward. Let \((A_1, A_2)\) be the actions of a zero-sum two-player game with utility functions \(u_1 : A_1 \times A_2 \rightarrow [-1, 1]\) and \(u_2 = -u_1\). We construct a three-player game \((A'_1, A'_2, A'_3, u'_1, u'_2, u'_3)\) in the following way:

- \(A'_1 = A_1 \cup \{\$\}\), \(A'_2 = A_2 \cup \{\$\}\) and \(A'_3 = \{\$\}\) (where $\$\$ is a fresh action).
- Let \(a'_2\) be an arbitrary action in \(A'_2 - \{\$\}\). We define the utility function \(u'_1\) to be

\[
u'_1(a_1, a_2, a_3) = \begin{cases} 
    u_1(a_1, a_2) & \text{if } a_1 \neq \$ \text{ and } a_2 \neq \$ \\
    u_1(a_1, a'_2) & \text{if } a_1 \neq \$ \text{ and } a_2 = \$
    0 & \text{otherwise (if } a_1 = \$ \text{)}
\end{cases}
\]

We define \(u'_2 = -u'_1\), and

\[
u'_3(a_1, a_2, a_3) = \begin{cases} 
    1 & \text{if } a_1 = \$
    -1 & \text{otherwise (if } a_1 \neq \$ \text{)}
\end{cases}
\]

The reader can verify that if player 1 has a strategy to assure utility greater than 0 in the zero-sum game, then in any profile of equilibrium in the three-player game he will play $\$ only for a negligible number of rounds, and the social welfare of the equilibrium will be \(-1\). On the other hand, if player 1 cannot assure utility at least 0, then a profile of strategies in which all three players play $\$\$ forever is an equilibrium and its social welfare is 1. Hence, even for \(\epsilon = 1\) we cannot approximate the social welfare of the optimal equilibrium, unless mean-payoff games are in P.

\[\square\]

### 9.3.4 An FPTAS to compute an \(\epsilon\)-equilibrium that is \(\delta\)-optimal

In this subsection, we assume that the utilities of the players are scaled to rationals in \([-1, 1]\), and we will describe an algorithm that computes an \(\epsilon\)-equilibrium that is \(\delta\)-optimal (with respect to all \(\epsilon\)-equilibria) and runs in time complexity that is polynomial in the input size and in \(\frac{1}{\epsilon}\) and \(\frac{1}{\delta}\). Subsection 9.3.3 suggests that in order to compute a
δ-optimal ε-equilibrium we should approximate (by some value) the values of \( \nu_1, \ldots, \nu_k \) and then compute the optimal infinite path (with respect to the sum of utilities) that has utility for player \( i \) that is greater than the approximation of \( \nu_i \). However, this approach would not work, since the optimal social welfare is not a continuous function with respect to the values \( \nu_1, \ldots, \nu_k \).

We denote by \( \text{OPT}_\epsilon \) the social welfare of the optimal \( \epsilon \)-equilibrium. We base our solution on the next lemma, which gives two key properties of \( \text{OPT}_\epsilon \).

**Lemma 98.**

1. If \( \epsilon_1 \geq \epsilon_2 \), then \( \text{OPT}_{\epsilon_1} \geq \text{OPT}_{\epsilon_2} \).

2. For every \( \alpha \in [0, 1] \) and \( \epsilon_1, \epsilon_2 > 0 \), let \( \epsilon = \alpha \epsilon_1 + (1 - \alpha) \epsilon_2 \), then there exists an \( \epsilon \)-equilibrium with social welfare \( \alpha \text{OPT}_{\epsilon_1} + (1 - \alpha) \text{OPT}_{\epsilon_2} \).

**Proof.** The first item of the lemma is a trivial observation. In order to prove the second item, we observe that by Lemma 93 (and since by Proposition 6 it is enough to consider only pure equilibria) it is enough to prove that there is an infinite path \( \pi \) with utility at least \( \nu_i - \epsilon \) in every dimension and with social welfare \( \alpha \text{OPT}_{\epsilon_1} + (1 - \alpha) \text{OPT}_{\epsilon_2} \). By Proposition 6, it is enough to show that there is a profile \( \sigma \) of mixed strategies (that need not be an equilibrium) that has a utility at least \( \nu - \epsilon \) in every dimension and has a social welfare at least \( \alpha \text{OPT}_{\epsilon_1} + (1 - \alpha) \text{OPT}_{\epsilon_2} \). The construction of \( \sigma \) is trivial. For \( i = 1, 2 \), let \( \sigma_{\epsilon_i} \) be a profile of strategies that induces an \( \epsilon_i \)-optimal equilibrium, then we construct \( \sigma \) by playing according to \( \sigma_{\epsilon_1} \) with probability \( \alpha \) and playing according to \( \sigma_{\epsilon_2} \) with probability \( 1 - \alpha \).

\[ \square \]

**Corollary 6.** For every \( \zeta \leq \frac{\delta}{4k} \) we have \( \text{OPT}_{\epsilon + \zeta} - \frac{\delta}{2} \leq \text{OPT}_\epsilon \leq \text{OPT}_{\epsilon + \zeta} \).

**Proof.** The fact that \( \text{OPT}_\epsilon \leq \text{OPT}_{\epsilon + \zeta} \) follows immediately from Lemma 98(1). To prove that \( \text{OPT}_{\epsilon + \zeta} - \delta \leq \text{OPT}_\epsilon \), we set \( \alpha = \frac{\zeta}{\epsilon} \) and by Lemma 98(2), and since \( \alpha \zeta + (1 - \alpha)(\epsilon + \zeta) = \epsilon \) we get

\[
\alpha \text{OPT}_\zeta + (1 - \alpha) \text{OPT}_{\epsilon + \zeta} \leq \text{OPT}_\epsilon
\]

since the utility function is scaled to \([-1, 1]\) we get that \( \text{OPT}_\zeta \geq -k \) and \( \text{OPT}_{\epsilon + \zeta} \leq k \), and thus we have

\[
-2k\alpha + \text{OPT}_{\epsilon + \zeta} \leq \text{OPT}_\epsilon
\]

since \( \alpha = \frac{\zeta}{\epsilon} \) we get

\[
-2k\frac{\zeta}{\epsilon} + \text{OPT}_{\epsilon + \zeta} \leq \text{OPT}_\epsilon
\]

and since \( \zeta \leq \frac{\delta}{4k} \) we have

\[
\text{OPT}_{\epsilon + \zeta} - \frac{\delta}{2} \leq \text{OPT}_\epsilon
\]

\[ \square \]
By the above corollary, to approximate $OPT_\epsilon$, it is enough to approximate $OPT_\epsilon + \zeta$ for some $\zeta \leq \frac{\epsilon}{4k}$. For this purpose, we extend the notion of $\epsilon$-equilibrium also for $k$-dimensional vectors, and we say that a profile of strategies is a $\beta$-equilibrium if player $i$ cannot improve its utility by at least $\beta_i$. Let us denote by $\min \beta$ and by $\max \beta$ the minimal and maximal element of $\beta$ (respectively). Then by definition, $OPT_{\min \beta} \leq OPT_\beta \leq OPT_{\max \beta}$, and by Corollary 6 we get that $OPT_\beta \leq OPT_{\max \beta} \leq OPT_{\beta} + (\max \beta - \min \beta) \cdot \frac{4k}{\epsilon}$.

We are now ready to present an FPTAS that computes a $\delta$-approximation for $OPT_\epsilon$:

1. Set $\zeta = \frac{\epsilon}{4k}$, and compute a $\zeta$ approximation of $\nu_i$ for every player $i$, and denote it by $r_i$. (2) Compute the optimal path $\pi$ (with respect to social welfare) that has utility at least $r_i - (\epsilon - \zeta)$ for every player, and return its social welfare.

We note that we can execute the first step of the algorithm in polynomial time due to (Observation 3.1), and we can execute the second step in polynomial time by solving the linear programming problem that we described in the proof of Lemma 96.

The next lemma proves the correctness of our approximation algorithm.

**Lemma 99.** Let $S(\pi)$ be the social welfare of $\pi$. Then $S(\pi) - \delta \leq OPT_\epsilon \leq S(\pi)$

**Proof.** We denote player-$i$ utility according to $\pi$ by $u_i(\pi)$ and we construct the vector $\beta$ by defining

$$\beta_i = \begin{cases} \nu_i - u_i(\pi) & \text{if } \nu_i - u_i(\pi) \geq (\epsilon - \zeta) \\ \epsilon - \zeta & \text{otherwise} \end{cases}$$

and we claim that $\pi$ is an optimal $\beta$-equilibrium. The claim holds because for every $\beta$-equilibrium, the utility of player $i$ is at least $\nu_i - (\epsilon - \zeta)$, and $\pi$ is the optimal path with utility at least $\nu_i - (\epsilon - \zeta)$ for every player. In addition, we claim that $\max \beta - \min \beta$ is at most $\frac{\epsilon}{2}$. The claim holds because by the construction of $\beta$ we have that $\min \beta \geq \epsilon - \frac{\epsilon}{2}$, and $\nu_i - u_i(\pi)$ is at most $\epsilon - \zeta$ (since $\pi$ is an $(\epsilon - \zeta)$-equilibrium). Hence $\max \beta - \min \beta \leq \frac{\epsilon}{2}$. Therefore, since $\max \beta \leq \epsilon$ and since

$$OPT_\beta \leq OPT_{\max \beta} \leq OPT_\beta + (\max \beta - \min \beta) \cdot \frac{4k}{\epsilon}$$

we get that

$$S(\pi) \leq OPT_\epsilon \leq S(\pi) + \frac{\epsilon}{2} \cdot \frac{4k}{\epsilon \delta}$$

and since $\zeta = \frac{\epsilon}{4k}$ we get that

$$S(\pi) \leq OPT_\epsilon \leq S(\pi) + \frac{\delta}{2}$$

and the assertion of the lemma follows.

Lemma 99 along with the complexity analysis that we provided, proves that there is an FPTAS to compute an $\epsilon$-equilibrium that is $\delta$-optimal, and Theorem 27(4) follows.
We also note that our proof for Theorem 27(4) gives a constructive (and polynomial) algorithm that computes a description of an actual $\epsilon$-equilibrium that is $\delta$-optimal.
Chapter 10

Quantitative Interprocedural Analysis

Static and interprocedural analysis. Static analysis techniques provide ways to obtain information about programs without actually running them on specific inputs. Static analysis explores the program behavior for all possible inputs and all possible executions. For non-trivial programs, it is impossible to explore all the possibilities, and hence static analysis uses approximations (abstract interpretations) to account for all the possibilities [42]. Static analysis algorithms generally operate on the interprocedural control-flow graphs (for brevity ICFGs). An ICFG consists of a collection of control-flow graphs (CFGs), one for each procedure of the program. The CFG of each procedure has a unique entry node and a unique exit node, and other nodes represent statements of the program and conditions (in other words, basic blocks of the program). In addition, there are call and return nodes for each procedure which represent invoking of procedures and return from procedures. Call-transitions connect call nodes to entry nodes; and return-transitions connect exit nodes to return nodes. Algorithmic analysis of ICFGs provides the mathematical framework for static analysis of programs. Interprocedural analysis with objectives such as reachability, set-based information, etc have been extensively studied in the literature [22, 66, 74, 81–83].

Quantitative objectives. A qualitative (or Boolean) objective assigns to every run of a program a Boolean value (accept or reject). A quantitative objective assigns to every run of a program a real value that represents a quality measure of the run. The analysis of programs with quantitative objectives is gaining huge prominence due to embedded systems with requirements on resource consumption, promptness of responses, performance analysis etc. Quantitative objectives have been proposed in several applications such as for worst-case execution time (see [104] for survey), power consumption [92], prediction of cache behavior for timing analysis [5], performance measures [16, 34, 51], to name a few. Another important feature of quantitative objectives is that they are very well-suited for anytime algorithms [19] (where anytime algorithms generate imprecise answers quickly, and proceed to construct progressively better approximate solutions with refinements) (for a more elaborate discussion see [30]).

Mean-payoff and ratio objectives. One of the most well-studied and mathematically elegant quantitative objectives is the mean-payoff objective, where a rational-valued weight is associated with every transition and the goal is to ensure that the
long-run average of the weights along a run is at least a given threshold \([14, 47, 110]\). For example, consider a weight function that assigns to every transition the resource (such as power) consumption, then the mean-payoff objective measures the average resource consumption along a run. Along with IC\(F\)Gs with mean-payoff objectives, we also consider ratio objectives. For ratio objectives, the transitions (events) of the IC\(F\)Gs are labelled as good, bad, or neutral events, and a positive weight function assigns a positive integer-valued weight to every transition, and the weight function represents how good or bad an event is. The quantitative analysis problem asks if there is a run of the program such that the ratio of the sum of the weights of the good events versus the weights of the bad events in the long-run is at least a given threshold. For example, consider a weight function that assigns weight 1 to each transition, and a labeling of events as follows: whenever a request is made is a bad event, whenever a request is pending is a good event, and whenever no request is pending is a neutral event. The ratio objective assigns the long-run average time between requests and the corresponding grant per request for a run, and measures timeliness of responses to requests. Finite-state systems (or intraprocedural finite-state programs) with mean-payoff objectives have been studied in the literature in \([47, 51, 70, 110]\) for performance modeling, and more recently applied in synthesis of reactive systems with quality guarantee \([16]\) and robustness \([17]\), reliability requirements and resource bounds of reactive systems \([21, 34, 44]\).

**Interprocedural quantitative analysis.** Quantitative objectives such as mean-payoff and ratio objectives provide the appropriate framework to express several important system properties such as resource consumption and timeliness. While finite-state systems with mean-payoff objectives have been studied in the literature, the static analysis of IC\(F\)Gs with mean-payoff and ratio objectives has largely been ignored. An interprocedural analysis is precise if it provides the meet-over-all-valid-paths solution (a path is valid if it respects the fact that when a procedure finishes it returns to the site of the most recent call). In the quantitative setting, the problem corresponds to finding the maximal value over all valid paths and to produce a witness (symbolic) path for that value. In this work, we consider precise interprocedural quantitative analysis for IC\(F\)Gs with mean-payoff and ratio objectives.

**Our contributions.** In this chapter we present a flexible and general modelling framework for quantitative analysis and show how it can be used to reason about quantitative properties of programs and about potential optimizations in the program. We present an efficient polynomial-time algorithm for precise interprocedural quantitative analysis, which is implemented as a tool. We demonstrate the efficiency of the algorithm with two case studies, and show that our approach scales to programs with thousands of methods.

1. *(Theoretical modeling).* We show that IC\(F\)Gs with mean-payoff and ratio objectives provide a robust framework that naturally captures a wide variety of static program analysis optimization and reasoning problems.
(a) (Detecting container usage). An exceedingly important problem for performance analysis is detection of runtime bloat that significantly degrades the performance and scalability of programs [72, 108]. A common source of bloat is inefficient use of containers [108]. We show that the problem of detecting usage of containers can be modeled as ICFGs with ratio objectives. A good use of a container corresponds to a good event and no use of the container is a bad event, and a misuse is represented as a low ratio of good vs bad events. Hence the container usage problem is naturally modeled as ratio analysis of ICFGs. While the problem of detecting container usage was already considered in [108], our different approach has the following benefits (see Section 10.4.2 for a comparison). First, our approach can handle recursion ([108] does not handle recursion). Second, our approach is sound, and does not yield false positives. Third, the approach of [108] ignored DELETE operations and we are able to take into consideration both ADD and DELETE operations (thus provide a more refined analysis). Moreover, our algorithmic approach for analysis of ICFGs is polynomial, whereas the algorithmic approach of [108] in the worst case can be exponential.

(b) (Static profiling of programs). We use our framework to model a conceptually new way for static profiling of programs for performance analysis. A line in the code (or a code segment) is referred as hot if there exists a run of the program where the line of code is frequently executed. For example, a function is referred as hot if there exists a run of the program where the function is frequently invoked, i.e., the frequency of calls to the function among all function calls is at least a given threshold. Again this problem is naturally modeled as ratio problem for ICFGs, and our approach statically detects methods that are more frequently invoked. Optimization of frequently executed code would naturally lead to performance improvements and reasoning about hot spots in the code can assist the compiler to apply optimization such as function inlining and loop unrolling (see Subsection 10.2.2 for more details).

(c) (Other applications). We show the generality of our framework by demonstrating that it provides an appropriate framework for theoretical modeling of diverse applications such as interprocedural worst-case execution time analysis, evaluating speedup in parallel computation, and interprocedural average energy consumption analysis.

2. (Algorithmic analysis). The quantitative analysis of ICFGs with mean-payoff objectives can be achieved in polynomial time by a reduction to pushdown systems with mean-payoff objectives, which can be solved in polynomial time (see Chapter 7). However, the resulting algorithm in the worst case has time complexity that is a polynomial of degree thirteen and space complexity that is a polynomial of degree six (which is prohibitive in practice). We exploit the special theoretical
properties of ICFGs in order to improve the theoretical upper bound and get an algorithm that in the worst case runs in cubic time and with quadratic space complexity. In addition, we exploit the properties of real-world programs and introduce optimizations that give a practical algorithm that is much faster than the theoretical upper bound when the relevant parameters (the total number of entry, exit, call, and returns nodes) are small, which is typical in most applications. Finally we present a linear reduction of the quantitative analysis problem with ratio objectives to mean-payoff objectives.

3. (Tool and experimental results). We have implemented our algorithm and developed a tool in the Java Soot framework [95]. We show through two case studies that our approach scales to relatively large programs from well-known benchmarks. The details of the case studies are as follows:

(a) (Detecting container usage). Our experimental results show that our tool scales to relatively large benchmarks (DaCapo 2009 [15]), and discover relevant and useful information that can be used to optimize performance of the programs. Our tool could analyze all containers in several benchmarks (like muffin) whereas [108] could analyze them partially (in muffin only half of the containers were analyzed in [108] — before the predefined time bound was exceeded). Our sound approach allows us to avoid false reports (that were reported by [108]) and our simple mathematical modelling even allows us to report misuses that were not reported by [108].

(b) (Static profiling of programs). We run an analysis to detect hot methods for various thresholds. Our experimental results on the benchmarks report only a small fraction of the functions as hot for high threshold values, and thus give useful information about potential functions to be optimized for performance gain. In addition we perform a dynamic profiling and mark the top 5% of the most frequently invoked functions as hot. Our experiments show a significant correlation between the results of the static and dynamic analysis. In addition, we show that the sensitivity and specificity of the static classification can be controlled by considering different thresholds, where lower thresholds increase the sensitivity and higher thresholds increase the specificity. We investigate the trade-off curve (ROC curve) and demonstrate the prediction power of our approach.

Thus we show that several conceptually different problems related to program optimizations are naturally modeled in our framework, and demonstrate that we present a flexible and generic framework for quantitative analysis of programs. Moreover, our case studies show that our tool scales to benchmarks with real-world programs.

Bibliographic note. The results of this chapter were first published in: Krishnendu Chatterjee, Andreas Pavlogiannis and Yaron Velner: Quantitative Interprocedural Analysis. POPL 2015.
10.1 Definitions

In this section we present formal definitions of interprocedural control-flow graphs, and the quantitative analysis problems. We will use an example program, described in Figure 10.1 and Figure 10.2, to demonstrate each definition.

Interprocedural control-flow graphs (ICFGs). A program $P$ with $m$ methods is modeled by an interprocedural control-flow graph (ICFG) $A$ which consists of a tuple $(A_1, \ldots, A_m)$ of $m$ modules, where each module $A_i = (N_i, En_i, Ex_i, Calls_i, Retns_i, \delta_i)$ represents a method (or the control-flow graph of a method) in the program. A module $A_i$ contains the following components:

- A finite set of nodes $N_i$.
- An entry node $En_i$, which represents the first node of the method.
- An exit node $Ex_i$, which represents termination of the method.
- A finite set $Calls_i$ that denotes the set of calls of the method, and a finite set $Retns_i$ that denotes the set of returns.
- A transition relation $\delta_i$ defined as follows: A transition in $A_i$ is either (i) between two nodes in the same module (internal transitions) or between a return node and a node in the same module (i.e., $u, v$ such that $u \in N_i \cup Retns_i$ and $v \in N_i \cup Calls_i$); or (ii) between a call node of a module $A_i$ and the entry node of a module $A_j$ (which models the invocation of method $j$ from method $i$); or (iii) between the exit node of module $A_i$ and a return node of a module $A_j$ (which models the case that method $i$ terminated and the run of the program continues in method $j$ which invoked method $i$).

We denote by $N = \bigcup_{1 \leq i \leq m} N_i$ and similarly, $En = \bigcup_{1 \leq i \leq m} \{En_i\}$; $Ex = \bigcup_{1 \leq i \leq m} \{Ex_i\}$; $Calls = \bigcup_{1 \leq i \leq m} Calls_i$; $Retns = \bigcup_{1 \leq i \leq m} Retns_i$; and $\delta = \bigcup_{1 \leq i \leq m} \delta_i$. In the sequel we use $N$ (resp. $N_i$) to denote all nodes in $A$ (resp. $A_i$), and refer to the nodes of $N \setminus (Calls \cup Retns \cup En \cup Ex)$ as internal nodes.

Quantitative ICFGs. A quantitative ICFG (QICFG for brevity) consists of an ICFG $A$ and a weight function $w$ that assigns a rational-valued weight $w(e) \in \mathbb{Q}$ to every transition $e$, where $\mathbb{Q}$ is the set of all rationals.

Example 10. Consider an example program shown in Figure 10.1 and Figure 10.2. In this example, (i) Modules: $A_1 = \text{main}, A_2 = \text{foo}$; (ii) Nodes: $N_2 = \{f:Entry, if(x>1), f:call foo, f:foo ret, x++, f:Exit\}$; (iii) Entry and exit: $En_2 = f:Entry$ and $Ex_2 = f:Exit$; (iv) Calls and returns: $Calls_2 = \{f:call foo\}$ and $Retns_2 = \{f:foo ret\}$; and (v) Transition: for example, $(x++, f:Exit) \in \delta_2$, $(f:Exit, f:foo ret) \in \delta_2$ and $(f:Exit, m:foo ret) \in \delta_2$.

Configurations and paths. A configuration consists of a sequence $c = (r_1, \ldots, r_j, u)$, where each $r_i$ is a return node (i.e., $r_i \in Retns$) and $u \in N_i$ is a node in one of the modules. Intuitively, when the module that $u$ belongs to terminates, the program will continue in $r_j$. A sequence of configurations is valid if it does not violate the transition relation, and a path $\pi$ is a valid sequence of configurations. We note that a path can
void main() {
    while(x) {
        if(y > 0) foo(x);
        else z = 7;
    }
    z++;
    return;
}

int foo(int x) {
    if(x > 1) foo(x - 1);
    x++;
    return x;
}

Figure 10.1: main

Figure 10.2: foo

be equivalently represented by the first configuration and a sequence of transitions. For a path \( \pi = \langle c_1, c_2, \ldots, c_\ell \rangle \) we denote by (i) \( n_i \): the node of configuration \( c_i \) (i.e., \( c_i = (r_1, \ldots, r_j, n_i) \)), and (ii) \( \alpha_i \): the stack string of \( c_i \) (i.e., \( \alpha_i = r_1, \ldots, r_j \)). For a path \( \pi \) let \( \pi[1, i] \) denote the prefix of length \( i \) of \( \pi \). A run of the program is modeled by a path in \( \mathcal{A} \).

**Example 11.** Consider the program and the corresponding ICFG shown in Figure 1 and Figure 2. An example of a run in the program modeled as a path in the ICFG is as follows: \( (\epsilon, m:\text{Entry}), (\epsilon, \text{while}(x)), (\epsilon, \text{if}(y > 0)), (\epsilon, m:\text{call foo}), (m:\text{foo ret}, f:\text{Entry}), (m:\text{foo ret}, \text{if}(x > 1)), (m:\text{foo ret}, f:\text{call foo}), (m:\text{foo ret}, f:\text{foo ret}, f:\text{Entry}), \ldots) \), where \( \epsilon \) denotes the empty stack.

**Ratio analysis problem.** In this work we consider the ratio analysis problem, where every transition of a ICFG has a label from the set \{good, bad, neutral\}. Intuitively, desirable events are labelled as good, undesirable events are labelled as bad, and other events are labelled as neutral. The ratio analysis problem, given a ICFG, a labeling of the events, and a threshold \( \lambda > 0 \), asks to determine whether there is a run where the ratio of sum of weights of good events vs the sum of weights of the bad events that is greater than the threshold \( \lambda \). Formally, we consider a positive integer-weight function \( w \), that assigns a positive integer-valued weight to every transition, and for good and bad events the weight denotes how good or how bad the respective event is. For a finite path \( \pi \) we denote by \( \text{good}(w(\pi)) \) (resp., \( \text{bad}(w(\pi)) \)) the sum of weights of the good (resp., bad) events in \( \pi \). In particular, for the weight function \( w \) that assigns weight 1 to every transition, we have that \( \text{good}(w(\pi)) \) (resp. \( \text{bad}(w(\pi)) \)) represents the number of good (resp. bad) events. We denote \( \text{Rat}(w(\pi)) = \frac{\text{good}(w(\pi))}{\max\{1, \text{bad}(w(\pi))\}} \) the ratio
of the sum of weights of good and bad events in \( \pi \) (note that in denominator we have \( \max\{1, \text{bad}(w(\pi))\} \) to remove the pathological case of division by zero). For an infinite path \( \pi \) we denote

\[
\text{LimRat}(w(\pi)) = \begin{cases} 
\liminf_{i \to \infty} \text{Rat}(w([1,i])) & \text{if } \pi \text{ has infinitely many } \\
0 & \text{good or bad events;}
\end{cases}
\]

informally this represents the ratio as the number of relevant (good/bad) events goes to infinity. Our analysis focuses on paths with unbounded number of relevant events, and infinitely many events provide an elegant abstraction for unboundedness. Hence, we investigate the following problem:

Given a \( ICFG \) with labeling of good, bad, and neutral events, a positive integer weight function \( w \), and a threshold \( \lambda \in \mathbb{Q} \) such that \( \lambda > 0 \), determine whether there exists an infinite path \( \pi \) such that \( \text{LimRat}(w(\pi)) > \lambda \).

Remark 6. Our approach can be extended to reason about finite runs by adding an auxiliary transition, labeled as a neutral event, from the final state of the program to its initial state (also see Section 10.2.3).

Mean-payoff analysis problem. In the mean-payoff analysis problem we consider a \( QICFG \) with a rational-valued weight function \( w \). For a finite path \( \pi \) in a \( QICFG \) we denote by \( w(\pi) \) the total weight of the path (i.e., the sum of the weights of the transitions in \( \pi \)), and by \( \text{Avg}(w(\pi)) = \frac{w(\pi)}{|\pi|} \) the average of the weights, where \( |\pi| \) denotes the length of \( \pi \). For an infinite path \( \pi \), we denote \( \text{LimAvg}(w(\pi)) = \liminf_{i \to \infty} \text{Avg}(w(\pi[1,i])) \). The mean-payoff analysis problem asks whether there exists an infinite path \( \pi \) such that \( \text{LimAvg}(w(\pi)) > 0 \). In Section 10.3 we show how the ratio analysis problem of \( ICFGs \) reduces to the mean-payoff analysis problem of \( QICFGs \).

Assigning context-dependent and path-dependent weights. In our model the numerical weights are assigned to every transition of a \( ICFG \). First, note that since we consider weight functions as an input and allow all weight functions, the weights could be assigned in a dependent way. Second, in general, we can have an \( ICFG \), and a finite-state deterministic automaton (such as a deterministic mean-payoff automaton [34]) that assigns weights. The deterministic automaton can assign weights depending on different contexts (or call strings) of invocations, or even independent of the context but dependent on the past few transitions (i.e., path-dependent), i.e., the automaton has the stack alphabet and transition of the \( ICFG \) as input alphabet and assigns weights depending on the current state of the automaton and an input letter. We call such a weight function regular weight function. Given a regular weight function specified by an automaton \( A \) and an \( ICFG \) we can obtain an \( ICFG \) (that represents the path-dependent weights) by taking their synchronous product, and hence we will focus on \( ICFGs \) for
algorithmic analysis. The regular weight function can also be an abstraction of the real weight function, e.g., the regular weight function is an over-approximation if the weights that it assigns to the good (resp. bad) events are higher (resp. lower) than the real weights. If the original weight function is bounded, then an over-approximation with a regular weight function can be obtained (which can be refined to be more precise by allowing more states in the automaton of the regular weight function). Note that the new ICFG which is obtained from an ICFG and automaton $A$ has a blowup in the number of states of $A$, and thus there is a tradeoff between the precision of $A$ and the size of the new ICFG constructed.

### 10.2 Applications: Theoretical Modeling

In this section we show that many diverse problems for static analysis can be reduced to ratio analysis of ICFGs. We will present experimental results (in Section 10.4) for the problems described in Section 10.2.1 and Section 10.2.2.

#### 10.2.1 Container analysis

The inefficient use of containers is the cause of many performance issues in Java. An excellent exposition of the problem with several practical motivations is presented in [108]. The importance of accurate identification of misuse of containers that minimizes (and ideally eliminates) the number of false warnings was emphasized in [108] and much effort was spent to avoid false warnings for real-world programs. We show that the ratio analysis for ICFGs provides a mathematically sound approach for the identification of inefficient use of containers.

**Two misuses.** We aim to capture two common misuses of containers following the definitions in [108]. The first inefficient use is an underutilized container that always holds very few number of elements. The cause of inefficiency is two-fold: (i) a container is typically created with a default number of slots, and much more memory is allocated than needed; and (ii) the functionality that is associated with the container is typically not specialized to the case that it has only very few elements. The second inefficiency is caused by overpopulated containers that are looked up rarely, though potentially they can have many elements. This causes a memory waste and performance penalty for every lookup. Thus we consider the following two cases of misuse:

1. A container is underutilized if there exists a constant bound on the number of elements that it holds for all runs of the program.
2. For a threshold $\lambda$, a container is overpopulated if for all runs of the program the ratio of Get vs Add operations is less than $\lambda$.

We note that our approach is demand-driven (where users can specify to check the misuse of a specific container).

**Modeling.** The modeling of programs as ICFGs is standard. We describe how the weight function and the ratio analysis problem can model the problem of detecting misuses. We abstract the different container operations into Get, Add, and Delete operations. For this purpose we require the user to annotate the relevant class methods
by \textit{Get}, \textit{Add}, or \textit{Delete}; and by a weight function that corresponds to the number of \textit{Get}, \textit{Add}, or \textit{Delete} operations that the method does (typically this number is 1). For example, in the class \texttt{HashSet}, the \texttt{add} method is annotated by \textit{Add}, the \texttt{contains} method is annotated by \textit{Get} and the \texttt{remove} operation is annotated by \textit{Delete}. The \texttt{clear} operation which removes all elements from the set is annotated by \textit{Delete} but with a large weight (if \texttt{clear} appears in a loop, it dominates the \textit{add} operations of the loop). We note that the annotation can be automated with the approach that is described in [108].

1. When detecting underutilized containers we define \textit{Add} operations as good events and \textit{Delete} operations as bad events, and check for threshold 1. Note that the relevant threshold is 1: if the (long-run) ratio of \textit{Add} vs \textit{Delete} is not greater than 1, then the total number of elements in the container is bounded by a constant.

2. When detecting overpopulated containers we define \textit{Get} operations as good events and \textit{Add} operations as bad events, and check for the given threshold $\lambda$.

In addition, since we wish to analyze heap objects, the allocation of the container is a bad event with a large weight (i.e., similar effect as of \texttt{clear}); see Example 12. The container is misused iff the answer to the ratio analysis problem is NO (note that in the problem description for container analysis we have quantification over all paths and for ratio analysis of ICFGs the quantification is existential). The detection is demand-driven and done for an allocated container $c$.

\textit{Details of modeling.} Intuitively, a transition in the call graph is good if it invokes a functionality that is annotated by a good operation (i.e., \textit{Add} operation for the underutilized analysis and \textit{Get} operation for the overpopulated analysis) and the object that invokes the operation points to container $c$, and it is bad if the invoked operation is annotated as bad. Formally, for a given allocated container $c$: If at a certain line a variable $t$ that \textit{may point to} $c$ invokes a good functionality, then we denote the transition as good. If $t$ \textit{must point to} $c$ and invokes a bad functionality, then we denote it as a bad events. All other transitions are neutral. Note that our modeling is conservative. The misuse is detected for the container $c$ if all runs of the program have a ratio of good vs bad events that is below the threshold (in other words, the container is not misused if there exists a path where the ratio of good vs bad events is above the threshold, and this exactly corresponds to the ratio analysis of ICFGs).

\textbf{Example 12.} We illustrate some important aspects of the container analysis problem with an example. Consider the program shown in Figure 10.3. We consider the containers that are allocated in line 9 and in line 20 and analyze them for underutilization. There exist runs that go through line 14 and properly use the container that is allocated in line 9, since \texttt{qux} method can add unbounded number of elements to the hash table (due to its recursive call). However, the container in line 20 is underutilized, since in every run the number of elements is bounded by 2. However, note that if the \texttt{Delete}
operation is not handled, then the container is reported as properly used. We note that since we assign large weights to the allocation of the container, this prevents the analysis from reporting that \( h_2 \) properly uses the container that is allocated in line 20. In summary, the example illustrates the following important features: (1) the proper usage of the container should be tested also outside of its allocation site\(^1\) (as opposed to the approach of [108]); (2) sometimes the proper usage of a container is due to recursion; and (3) handling DELETE operations appropriately increases the precision of analysis.

While these important features are illustrated with the toy example, such behaviors were also manifested in the programs of the benchmarks (see Subsection 10.4.2 for details).

Soundness. Our ratio analysis approach for ICFGs is both sound and complete (with respect to the weighted abstracted ICFG). Since we use a conservative approach for assigning bad and good events, the ICFG we obtain for the misuse analysis of containers is sound and we get the following result.

Theorem 28. (Soundness). The underutilized and overpopulated container analysis through the ratio analysis problem on ICFGs is sound (do not report false positives, i.e., any reported misused container is truly misused).

Remark 7. We remark about the significance of the soundness of our approach.

- The soundness criteria is a very important and desirable feature for container analysis (for details see [106, 108]), because a reported misused container needs to be analyzed manually and incurs a substantial effort for optimization. Hence as argued in [106, 108], spurious warnings (false positive) of misuse must be minimized. In our approach, a misuse is reported iff in every run a misuse is detected, and with a sound (over approximation) annotation of the weights our approach is sound.

- The soundness of our approach is with respect to a sound (over approximation) annotation of the ADD, DELETE and GET operations. In addition, for a given

\(^1\)E.g., an HashTable is allocated in bar function but the proper usage is done outside the allocation site, namely, after the termination of bar.
ICFG, our ratio analysis is precise (i.e., both sound and complete), hence, our container analysis is sound.

10.2.2 Static profiling

Finding the most frequently executed lines in the code can help the programmer to identify the critical parts of the program and focus on the optimization of these parts. It can also assist the compiler (e.g., a C compiler) to decide whether it should apply certain optimizations such as function inlining (replacing a function call by the body of the called function) and loop unrolling (re-write the loop as a repeated sequence of similar independent statements). These optimizations can reduce the running time of the program, but on the other hand, they increase the size of the (binary) code. Hence, knowing whether the function or loop is hot (frequently invoked) is important when considering the time vs. code size tradeoff. In this subsection we present the model for profiling the frequency of function calls (which allows finding hot functions), and we note that our profiling technique is generic and can be scaled to detect other hot spots in the code (e.g., hot loops).

Problem description. Given a program with several functions, a function \( f \) is called \( \lambda \)-hot, if there exists a run (of unbounded length) of the program where the frequency of calls to \( f \) (among all function calls in the run) is at least \( \lambda \). Formally, for a run, given a prefix of length \( i \), let \( \#f(i) \) denote the number of calls to \( f \) and \( \#c(i) \) denote the number of function calls in the prefix of length \( i \). The function is \( \lambda \)-hot if there exists a run such that \( \lim \inf_{i \to \infty} \frac{\#f(i)}{\#c(i)} > \lambda \).

Modeling. The modeling of programs as ICFGs is straightforward. We describe the labeling of events and weight function in ICFGs to determine if a function \( f \) is \( \lambda \)-hot. First we label call-transitions to \( f \) as good events and assign weight 1; then we label all other call-transitions as bad events and assign them weight 1. To ensure that the number of calls to \( f \) also appear in the denominator (in the total number of calls) we label transitions from the entry node of \( f \) as bad events with weight 1. The function \( f \) is \( \lambda \)-hot iff the answer to the ratio analysis problem with threshold \( \lambda \) is YES.

10.2.3 Estimating worst-case execution time

The approach of [30] for estimating worst-case execution time (WCET) is also naturally captured by ratio analysis. While the intraprocedural problem was considered in [30], our approach allows the more general interprocedural analysis. In this approach, we consider (as in [30]) that each program statement is assigned a cost that corresponds to its running time (e.g., number of CPU cycles).

Modeling. The modelling of WCET analysis of the program is as follows: We add to the ICFG of the program a transition from every terminal node to the initial node, and every such transition is a bad event with weight 1. All the other transitions are good events and their weight is their cost (running time). The WCET of the program is at most \( N \) cycles if and only if the answer to the ratio analysis problem with threshold \( N \) is NO.
10.2.4 Evaluating the speedup in a parallel computation

The speed of a parallel computing is limited by the time needed for the sequential fraction of the program. For example, if a program runs for 10 minutes on a single processor core, and a certain part of the program that takes 2 minutes to execute cannot be parallelized, then the minimum execution time cannot be less than two minutes (regardless of how many processors are devoted to a parallelized execution of this program). Hence, the speedup is at most 5. Amdahl’s law \[10\] states that the theoretical speedup that can be obtained by executing a given algorithm on a system capable of executing \(n\) threads of execution is at most \(\frac{1}{B + \frac{1}{B}(1-B)}\), where \(B\) is the fraction of the algorithm that is strictly serial. Our ratio analysis technique can be used to (conservatively) estimate the value of \(B\) and thus to evaluate the outcome of Amdahl’s law.

Modeling. As in Section 10.2.3, we consider that the cost of every program statement is given, and we add to the ICFG of the program a transition from every terminal node to the initial node, this time as a neutral event with weight 0. All the transitions of the code that cannot be parallelized are defined as good events, and the other transitions are defined as bad events. We denote by \(P\) the fraction of the code that can be parallelized and by \(S\) the fraction of the code that is strictly serial. The value of \(\frac{S}{P}\) is at most \(\lambda\) if and only if the answer to the ratio analysis problem with threshold \(\lambda\) is NO. Hence \(B\) is bounded by \(\frac{1}{1+\frac{1}{\lambda}}\) for which the answer to the ratio analysis problem with threshold \(\lambda\) is NO.

10.2.5 Average energy consumption

In the case of many consumer electronics devices, especially mobile phones, battery capacity is severely restricted due to constraints on size and weight of the device. This implies that managing energy well is paramount in such devices. Since most mobile applications are non-terminating (e.g., a web browser), the most important metric for measuring energy consumption is the average memory consumption per time unit \[28\], e.g., watts per second.

Modeling. We consider that the running time and energy consumption of each statement in the application code is given (or is approximated). In our modeling we split each transition in the ICFG into two consecutive transitions, the first is a good event and the next is a bad event. The good event is assigned with a weight that corresponds to the energy consumption of the program statement and the bad event is assigned with a weight that corresponds to the running time of the statement. The average energy consumption of the application is at most \(\lambda\) if and only if the answer to the ratio analysis problem is NO.

10.3 Algorithm for Quantitative Analysis of QICFGs

In this section we present three results. The mean-payoff analysis problem for QICFGs can be solved in polynomial time, this can be derived from Chapter 7. First, we present
an algorithm that significantly improves the current theoretical bound for the problem for $QICFG$s. Second, we present an efficient algorithm that in most practical cases is much faster as compared to the theoretical upper bound. Finally, we present a linear reduction of the ratio analysis problem to the mean-payoff analysis problem for $QICFG$s.

10.3.1 Improved algorithm for mean-payoff analysis

In this section we first discuss the basic polynomial-time algorithm for mean-payoff analysis of $QICFG$s that can be obtained from the results on pushdown systems shown in Chapter 7.

**Results of Chapter 7 and reduction.** The results of Chapter 7 show that pushdown systems with mean-payoff objectives can be solved in polynomial time. Given a pushdown system with state space $Q$ and stack alphabet $\Gamma$, the polynomial-time algorithm of Chapter 7 can be described as follows. The algorithm is iterative, and in each iteration it constructs a finite graph of size $O(|Q|^2 \cdot |\Gamma|^2)$ and runs a Bellman-Ford style algorithm on the finite graph from each vertex. The Bellman-Ford algorithm on the finite graph from all vertices in each iteration requires $O(|Q|^3 \cdot |\Gamma|^6)$ time and $O(|Q|^2 \cdot |\Gamma|^4)$ space. The number of iterations required is $O(|Q|^2 \cdot |\Gamma|^2)$. Thus the time and space requirement of the algorithm are $O(|Q|^5 \cdot |\Gamma|^8)$ and $O(|Q|^2 \cdot |\Gamma|^4)$, respectively. A $QICFG$ can be interpreted as a pushdown system where $N$ corresponds to $Q$ and $\text{Retns}$ corresponds to $\Gamma$.

**Theorem 29.** *(Basic algorithm from Chapter 7).* The mean-payoff analysis problem for $QICFG$s can be solved in $O(|N|^5 \cdot |\text{Retns}|^8)$ time and $O(|N|^2 \cdot |\text{Retns}|^4)$ space, respectively.

**Improved algorithm.** We will present an improved polynomial-time algorithm for the mean-payoff analysis of $QICFG$s. The improvement relies on the following properties of $QICFG$s:

1. The transitions of a module are independent of the stack of a configuration, while in pushdown systems the transitions can depend on the top symbol of the stack. This enables to reduce the size of the finite graphs to be considered in every iteration.

2. Every call node has only one corresponding return node. Therefore, if a module $A_1$ invokes a module $A_2$, then the behavior of $A_1$ after the termination of $A_2$ is independent of $A_2$. This enables us to reduce the number of iterations to $O(|\text{Calls}|)$.

To present the improved algorithm and its correctness formally, we need a refined analysis and extensions of the results of Chapter 7. We first describe a key aspect and present an overview of the solution.

**Remark 8.** *(Infinite-height lattice).* Our algorithm will be an iterative algorithm till some fixpoint is reached. However, for interprocedural analysis with finite-height lat-
tices, fixpoints are guaranteed to exist. Unfortunately in our case for mean-payoff objectives, it is an infinite-height lattice. Thus a fixpoint is not guaranteed. For this reason the analysis for mean-payoff objectives is more involved, and this is even in the case of finite graphs. For example, for reachability objectives in finite graphs linear-time algorithms exist, whereas for finite graphs with mean-payoff objectives the best-known algorithms (for over three decades) are quadratic [61].

Solution overview. In finite graphs the solution for the mean-payoff analysis is to check whether the graph has a cycle $C$ such that the sum of weights of $C$ is positive. If such a cycle exists, then a lasso path that leads to the cycle and then follows the cyclic path forever has positive mean-payoff value. For QICFGs we show that it is enough to find either a loop in the program such that the sum of weights of the loop is positive or a sequence of calls and returns with positive total weight such that the last invoked module is the same as the first invoked module. For this purpose we compute a summary function that finds the maximum weight (according to the sum of weights) path between every two statements of a method (i.e., between every two nodes of a module). The computation is an extension of the Bellman-Ford algorithm to QICFGs. We show that it is enough to compute a summary function for QICFGs with a stack height that is bounded by some constant, and then all that is left is to mark pairs of nodes such that the weight of a maximal weight path between them is unbounded. In finite graphs the maximum weight between two vertices is unbounded only if the graph has a cycle with positive sum of weights (i.e., a path with positive total weight that can be pumped). For QICFGs it is also possible to pump special types of acyclic paths. We first characterize these pumpable paths (up to Lemma 101). We then show how to compute a bounded summary function (Lemma 102 and the paragraph that follows it and Example 14). Finally we show how to use the summary function to solve the mean-payoff analysis problem. We start with the basic notions related to stack heights and pumpable paths, and their properties which are crucial for the algorithm.

Stack heights. The configuration stack height of $c = (r_1, \ldots, r_j, u)$, denoted as $\text{SH}(c)$, is $j$. For a finite path $\pi = ((\alpha_1, n_1), \ldots, (\alpha_\ell, n_\ell))$, the stack height of the path (denoted by $\text{SH}(\pi)$) is the maximal stack height of all the configurations in the path. Formally $\text{SH}(\pi) = \max\{|\alpha_1|, \ldots, |\alpha_\ell|\}$. The additional stack height of $\pi$ is the additional height of the stack in the segment of the path, i.e., the additional stack height $\text{ASH}(\pi)$ is $\text{SH}(\pi) - \max\{|\alpha_1|, |\alpha_\ell|\}$.

Pumpable pair of paths. Let $\pi = (c_1t_1t_2\ldots)$ be a finite or infinite path (where each $t_i$ is a transition in the QICFG). A pumpable pair of paths for $\pi$ is a pair of non-empty sequences of transitions: $(p_1, p_2) = (t_1, t_1+1, \ldots, t_1+t_\ell_1, t_2, t_2+1, \ldots, t_2+t_\ell_2, \ldots)$, for $\ell_1, \ell_2 \geq 0$, $i_1 \geq 0$ and $i_2 > i_1 + \ell_1$ such that for every $j \geq 0$ the path $\pi^j_{(p_1, p_2)}$ obtained by pumping the pair $p_1$ and $p_2$ of paths $j$ times each is a valid path, i.e., for every $j \geq 0$ we have

$$\pi^j_{(p_1, p_2)} = (c_1t_1\ldots t_{i_1-1}(p_1)^j t_{i_1+\ell_1+1} \ldots t_{i_2-1}(p_2)^j t_{i_2+\ell_2+1} \ldots)$$
is a valid path. We illustrate the above definitions with the next example.

Example 13. Consider the program from Figure 10.1 and Figure 10.2 and the corresponding ICFG. A possible path in the program is

\[
\begin{align*}
m : Entry & \rightarrow \text{while}(x) \rightarrow \text{if}(y > 0) \rightarrow m : call \ foo \rightarrow f : Entry \rightarrow \text{if}(x > 1) \rightarrow \\
& f : call \ foo \rightarrow f : Entry \rightarrow \text{if}(x > 1) \rightarrow x++ \rightarrow f : Exit \rightarrow f : foo \ ret \rightarrow x++ \rightarrow \\
& f : Exit \rightarrow m : foo \ ret \rightarrow \text{while}(x)
\end{align*}
\]

and we denote this path with \(\pi\). Then \(\text{ASH}(\pi) = 2\), and the pair of paths \(f : Entry \rightarrow \text{if}(x > 1) \rightarrow f : call \ foo \rightarrow f : foo \ ret \rightarrow x++ \rightarrow f : Exit\) is a pumpable pair of paths.

In the next lemmas we first show that every path with large additional stack has a pumpable pair of paths, and then establish the connection of additional stack height and the existence of pumpable pair of paths with positive weights in Lemma 101. The key intuition for the proof of the next lemma is that a path with \(\text{ASH}(\pi) > |\text{Calls}| + 1\) must contain a recursive call that can be pumped.

Lemma 100. Let \(\pi\) be a finite path with \(\text{ASH}(\pi) = d > |\text{Calls}| + 1\). Then \(\pi\) has a pumpable pair of paths.

Proof. Intuitively a path with \(\text{ASH}(\pi) > |\text{Calls}| + 1\) must contain a recursive call that can be pumped. We now present the detailed argument. Let \(c_0\) and \(c_k\) be the starting and the end configurations of the finite path \(\pi\), respectively. Let \(\ell = \max\{\text{SH}(c_0), \text{SH}(c_k)\}\).

Given \(\pi\), let \(c_1\) be the first configuration in \(\pi\) of stack height strictly greater than \(\ell\) and with a call node \(n_1 \in \text{Calls}_i\) (for some module \(A_i\)) such that there exists a configuration \(c_2\) in \(\pi\) with a call node \(n_2 \in \text{Calls}_i\) satisfying the following conditions: (i) \(n_1 = n_2\) and (ii) in the path segment in \(\pi\) between \(c_1\) and \(c_2\) the stack height is always at least \(\text{SH}(c_1)\). Moreover, let \(c_3\) be the first configuration after \(c_2\) of stack height \(\text{SH}(c_1)\) and with a return node \(n_3 \in \text{Retns}_i\). We first justify the existence of these configurations: (i) the existence of \(c_1\) and \(c_2\) follows by the pigeonhole principle and the fact that \(\text{ASH}(\pi) > |\text{Calls}| + 1\); and (ii) the existence of \(c_3\) follows because \(\text{SH}(c_1) > \text{SH}(c_k)\) and hence the call corresponding to \(c_1\) must return in the path \(\pi\). Note that existence of \(c_3\) (i.e., the return of the call of \(c_1\)) implies the existence of a configuration \(c_4\) with a return node \(n_4 \in \text{Retns}_i\) in the path such that \(\text{SH}(c_4) = \text{SH}(c_2)\), (this corresponds to the return of the call of \(c_2\)) Note that since \(n_1 = n_2\), it follows that \(n_3 = n_4\) (as they correspond to the return of the same call node). The path segment \(p_1\) of \(\pi\) between \(c_1\) and \(c_2\), and the path segment \(p_2\) of \(\pi\) between \(c_4\) and \(c_3\), constitutes a pumpable pair. The result follows.

Lemma 101. Let \(c_1, c_2\) be two configurations and \(j \in \mathbb{Z}\). Let \(d \in \mathbb{N}\) be the minimal additional stack height of all paths between \(c_1\) and \(c_2\) with total weight at least \(j\). If \(d > |\text{Calls}| + 1\), then there exists a path \(\pi^*\) from \(c_1\) to \(c_2\) with additional stack height \(d\) that has a pumpable pair \((p_1, p_2)\) with \(w(p_1) + w(p_2) > 0\).
Proof. Let us consider the set of paths \( \Pi \) between \( c_1 \) and \( c_2 \) with total weight at least \( j \), and let \( \Pi_{\text{min}} \) be the subset of \( \Pi \) that has minimal additional stack height. The proof is by induction on the length of paths in \( \Pi_{\text{min}} \). Consider a path \( \pi \) from \( \Pi_{\text{min}} \) that has the shortest length among all paths in \( \Pi_{\text{min}} \). Since \( \text{ASH}(\pi) = d > |\text{Calls}| + 1 \), then by Lemma 100 it contains a pumpable pair. Let us consider the path \( \pi_1 \) obtained from \( \pi \) by pumping the pumpable pair zero times (i.e., the pumpable pair is removed). Since we remove a part of the path we have that \( \text{ASH}(\pi_1) \leq \text{ASH}(\pi) \). If \( w(\pi_1) \geq w(\pi) \), then we obtain a path \( \pi_1 \) with weight at least \( j \), with either smaller additional stack height than \( \pi \), or of shorter length, contradicting that \( \pi \) is the shortest length minimal additional stack height path with weight at least \( j \). Hence we must have \( w(\pi_1) < w(\pi) \), and hence the pumpable pair has positive weight. Now for an arbitrary path \( \pi \) in \( \Pi_{\text{min}} \) we obtain that it has a pumpable pair. Either the pumpable pair has positive weight and we are done, else removing the pumpable pair we obtain a shorter length path of the same stack height, and the result follows by inductive hypothesis on the length of paths. 

Our algorithm for the mean-payoff analysis problem is based on detecting the existence of certain non-decreasing paths with positive weight. The maximal weights of such non-decreasing paths between node pairs are captured with the notion of a summary function and bounded summary functions (with bounded additional stack height). We now define them, and establish the lemma related to the number of bounded summary functions to be computed.

Local minimum and non-decreasing paths. A configuration \( c_i \) in a path \( \pi = \langle c_1, \ldots, c_\ell \rangle \) is a local minimum if the stack height of \( c_i \) is minimal in \( \pi \), i.e., \( |\alpha_i| = \min(|\alpha_1|, \ldots, |\alpha_\ell|) \). A path from configuration \((\alpha, n_1)\) to \((\alpha\beta, n_2)\) is a non-decreasing \( \alpha \)-path if \((\alpha, n_1)\) is a local minimum. Note that if a sequence of transitions is a non-decreasing \( \alpha \)-path for some \( \alpha \in \text{Retns}^* \), then the same sequence of transitions is a non-decreasing \( \gamma \)-path for every \( \gamma \in \text{Retns}^* \). Hence, we say that \( \pi \) is a non-decreasing path if there exists \( \alpha \in \text{Retns}^* \) such that \( \pi \) is a non-decreasing \( \alpha \)-path.

Summary function. Given the QICFG \( A \) and \( \alpha \in \text{Retns}^* \), we define a summary function \( s_\alpha : \bigcup_{1 \leq \ell \leq m} (N_\ell \times N_\ell) \to \{-\infty\} \cup \mathbb{Z} \cup \{\omega\} \) as:

1. \( s_\alpha(n_1, n_2) = z \in \mathbb{Z} \) iff the weight of the maximum weight non-decreasing path from configuration \((\alpha, n_1)\) to configuration \((\alpha, n_2)\) is \( z \).
2. \( s_\alpha(n_1, n_2) = \omega \) iff for all \( j \in \mathbb{N} \) there exists a non-decreasing path from \((\alpha, n_1)\) to \((\alpha, n_2)\) with weight at least \( j \).
3. \( s_\alpha(n_1, n_2) = -\infty \) iff there is no non-decreasing path from \((\alpha, n_1)\) to \((\alpha, n_2)\).

We note that for every \( \alpha, \beta \in \text{Retns}^* \) it holds that \( s_\alpha \equiv s_\beta \). Hence, we consider only \( s \equiv s_\epsilon \) (where \( \epsilon \) is the empty string and corresponds to empty stack). The computation of the summary function is done by considering stack height bounded summary functions defined below.

Stack height bounded summary function. For every \( d \in \mathbb{N} \), the stack height bounded
summary function $s_d : \bigcup_{i \leq \ell \leq m} (N_i \times N_i) \to \{-\infty\} \cup \mathbb{Z} \cup \{\omega\}$ is defined as follows: (i) $s_d(n_1, n_2) = z \in \mathbb{Z}$ iff the weight of the maximum weight non-decreasing path from $(\epsilon, n_1)$ to $(\epsilon, n_2)$ with additional stack height at most $d$ is $z$; (ii) $s_d(n_1, n_2) = \omega$ iff for all $j \in \mathbb{N}$ there exists a non-decreasing path from $(\epsilon, n_1)$ to $(\epsilon, n_2)$ with weight at least $j$ and additional stack height at most $d$; and (iii) $s_d(n_1, n_2) = -\infty$ iff there is no non-decreasing path with additional stack height at most $d$ from $(\epsilon, n_1)$ to $(\epsilon, n_2)$.

Facts of summary functions. We have the following facts: (i) for every $d \in \mathbb{N}$, we have $s_{d+1} \geq s_d$ (monotonicity); and (ii) $s_{d+1}$ is computable from $s_d$ and the QICFG. By the above facts we get that if $s_d \equiv s_{d+1}$, i.e., if a fix point is reached, then $s \equiv s_d$. For interprocedural analysis with finite-height lattices, fix points are guaranteed to exist. Unfortunately in our case, the image of $s_i$ is infinite and moreover, it is an infinite-height lattice. Thus a fix point is not guaranteed. The next lemma shows that we can compute all the non-$\omega$ values of $s$ with the bounded summary function.

**Lemma 102.** Let $d = |\text{Calls}| + 1$. For all $n_1, n_2 \in \mathbb{N}$, if $s(n_1, n_2) \in \mathbb{Z} \cup \{-\infty\}$, then $s(n_1, n_2) = s_d(n_1, n_2)$.

**Proof.** Obviously $s(n_1, n_2) \geq s_d(n_1, n_2)$. If $s_d(n_1, n_2) < s(n_1, n_2)$ it follows that there exists a non-decreasing path $\pi$ from $n_1$ to $n_2$ with $w(\pi) > s_d(n_1, n_2)$. By the definition of the bounded height summary function it follows that $\text{ASH}(\pi) > d$, and w.l.o.g we assume that $\pi$ has the minimal additional stack height among all non-decreasing paths from $n_1$ to $n_2$ with weight $w(\pi)$. Then by Lemma 101 it follows that $\pi$ has a pumpable pair $(p_1, p_2)$ with $w(p_1) + w(p_2) = w_p > 0$. Hence, for every $j \geq 0$ the path $\pi^j$ that is obtained from $\pi$ by pumping the pair $(p_1, p_2)$ exactly $j$ times has weight $w(\pi^j) = w(\pi) + (j - 1) \cdot w_p$, and it is a valid non-decreasing path from $n_1$ to $n_2$. Hence, for every $\ell \in \mathbb{N}$ the path $\pi^\ell$ for $j = \lceil \frac{\ell - w(\pi)}{w_p} \rceil + 1$ satisfies $w(\pi^j) \geq \ell$ (if $\ell \leq w(\pi)$, then we set $j = 1$). By definition we get that $s(n_1, n_2) = \omega$, and this completes the proof. \hfill \square

By Lemma 102 we get that if $s_{d+1}(n_1, n_2) > s_d(n_1, n_2)$ (for $d = |\text{Calls}| + 1$), then $s(n_1, n_2) = \omega$. Hence, the summary function $s$ is obtained by the fix point of the following computation: (i) Compute $s_{i+1}$ from $s_i$ up to $s_d$ for $d = |\text{Calls}| + 1$; (ii) for $i \geq |\text{Calls}| + 1$, if $s_{i+1}(n_1, n_2) > s_d(n_1, n_2)$, then set $s_{i+1}(n_1, n_2) = \omega$; (iii) a fix point is reached after at most $O(|\text{Calls}|)$ iterations (say $j$ iterations), and then we set $s \equiv s_j$. This establishes that we require only $O(|\text{Calls}|)$ iterations as compared to $O(|N|^2 \cdot |\text{Retns}|^2)$ iterations. The number of returns and calls are the same and thus we significantly improve the number of iterations required from the quartic worst-case bound to linear bound. We now describe the computation of every iteration to obtain $s_{i+1}$ from $s_i$.

**Computation of $s_{i+1}$ from $s_i$.** We first compute a partial function, namely, $s'_{i+1} : En \times Ex \to \{-\infty, \omega\} \cup \mathbb{Z}$ that satisfies $s'_{i+1}(n_1, n_2) = s_{i+1}(n_1, n_2)$ for every $n_1 \in En$ and $n_2 \in Ex$. We initialize $s'_0(n_1, n_2) = s_0(n_1, n_2)$. For every module $A_i$ we construct a finite graph $G'_i$ by taking all the nodes and transitions of $A_i$ and by adding a transition
between every call node and its corresponding return node. For every transition between a pair of nodes \( n_1, n_2 \in N_\ell \setminus (\text{Calls}_\ell \cup \text{Retns}_\ell) \) we assign the weight according to the original weight in \( \mathcal{A} \). For every transition between a call node that invokes module \( A_p \) and a corresponding return node we assign the weight \( s'_i(\ell) \). To compute \( s'_{i+1} \) for module \( A_\ell \) we run one Bellman-Ford iteration over \( G_\ell \) for source node \( E\ell \) and target node \( E\ell \). We observe the next two key properties of \( s' \):

- For every iteration \( i \), a module \( A_\ell \), and pair of nodes \( n_1, n_2 \in N_\ell \) we have that the weight of the maximum weight path between \( n_1 \) and \( n_2 \) in \( G_\ell \) is exactly \( s_{i+1}(n_1, n_2) \) (the proof is by a simple induction over \( i \)).

- If \( s'_{i+1} \equiv s'_i \), then \( s_{i+1} \equiv s_i \) (follows from the first key property).

Hence, to compute \( s \) we compute \( s'_{i+1} \) from \( s'_i \) until we get \( s'_{i+1} \equiv s'_i \), and then we compute all pairs maximum weight paths (e.g., by the Floyd-Warshall algorithm) over every \( G_\ell \) and get \( s_{i+1} \) (and \( s_{i+1} \equiv s \)). The Floyd-Warshall algorithm has a cubic time complexity and quadratic space complexity \([41]\). Therefore, the time complexity for computing every iteration of \( s_i \) is \( O(\sum |N_\ell|^2) \) and the complexity of the last step is \( O(\sum |N_\ell|^3) \). The space complexity of the last step is \( O(\max\{|N_1|, \ldots, |N_n|\}^2) \), but to store \( s_i \) we require \( O(\sum |N_\ell|^2) \) space.

**Summary graph.** Given QICFG \( \mathcal{A} \) with a summary function \( s \), we construct the summary graph \( \text{Gr}(\mathcal{A}) = (V, E) \) of \( \mathcal{A} \) with a weight function \( w : E \to \mathbb{Z} \cup \{\omega\} \) as follows: (i) \( V = N \setminus (E \cup \text{Retns}) \); and (ii) \( E = E_{\text{internal}} \cup E_{\text{calls}} \) where \( E_{\text{internal}} = \{(n_1, n_2) \mid n_1, n_2 \in N_\ell \text{ for some } \ell, \text{ and } s(n_1, n_2) > -\infty\} \) contains the transitions in the same module and \( E_{\text{calls}} = \{(n_1, n_2) \mid n_1 \in \text{Calls} \text{ and } n_2 \in E\ell \text{ and } n_1 \text{ is a call to a module with entry node } n_2\} \) contains the call transitions. The weights of \( E_{\text{internal}} \) are according to the summary function \( s \) and the weights of \( E_{\text{calls}} \) are according to the weights of these transitions in \( \mathcal{A} \) (i.e., according to \( w \)). A simple cycle in \( \text{Gr}(\mathcal{A}) \) is a **positive simple cycle** iff one of the following conditions hold: (i) the cycle contains an \( \omega \) edge; or (ii) the sum of the weights of the cycles according to the weights of the summary graph is positive. Lemma 103 shows the equivalence of the mean-payoff analysis problem and positive cycles in the summary graph.

**Lemma 103.** A QICFG \( \mathcal{A} \) has a path \( \pi \) with \( \text{LimAvg}(w(\pi)) > 0 \) iff the summary graph \( \text{Gr}(\mathcal{A}) \) has a (reachable) positive cycle.

**Proof.** If \( \text{Gr}(\mathcal{A}) \) does not contain a positive cycle, then it follows that the weight of every non-decreasing path in \( \mathcal{A} \) is bounded by the weight of the maximum weight path in \( \text{Gr}(\mathcal{A}) \). Hence, for every infinite path \( \pi \) we get that every prefix of \( \pi \) is a non-decreasing path from the initial configuration with bounded weight (sum of weights bounded from above), and therefore \( \text{LimAvg}(w(\pi)) \leq 0 \). Conversely, if \( \text{Gr}(\mathcal{A}) \) has a positive cycle, then it follows that there is a path \( \pi_0 \pi_1 \) in \( \text{Gr}(\mathcal{A}) \) such that \( \pi_0 \) and \( \pi_1 \) are non-decreasing paths, \( \pi_1 \) begins and ends in the same node (possibly at higher stack height) and \( w(\pi_1) > 0 \). Hence, the path \( \pi_0 \pi_1^\omega \) is a valid path and satisfies \( \text{LimAvg}(w(\pi_0 \pi_1^\omega)) = \frac{w(\pi_1)}{|\pi_1|} > 0 \), where \( \pi_1^\omega = \pi_1 \cdot \pi_1 \cdot \pi_1 \ldots \) is the infinite concatenation of the finite path \( \pi_1 \). The desired result follows. \( \square \)
Algorithm 3 Mean-payoff QICFG Analysis

1: for $\ell \leftarrow 1$ to $m$ do
2: \[ s'_0(En_{\ell}, Ex_{\ell}) \leftarrow \text{BELLMAN-FORD}(A_{\ell}) \{ \text{Compute } s'_0 \text{ by running Bellman-Ford algorithm on } A_{\ell} \} \]
3: end for
4: $i \leftarrow 1$
5: loop
6: for $\ell \leftarrow 1$ to $m$ do
7: Construct $G_{i-1}^{\ell}$ according to $s'_{i-1}$
8: \[ s'_i(En_{\ell}, Ex_{\ell}) \leftarrow \text{BELLMAN-FORD}(G_{i-1}^{\ell}) \{ \text{Compute } s'_i \text{ by running Bellman-Ford algorithm over } G_{i-1}^{\ell} \} \]
9: end for
10: if $s'_i \equiv s'_{i-1}$ then
11: break
12: end if
13: if $i > |\text{Calls}| + 1$ then
14: for $\ell \leftarrow 1$ to $m$ do
15: if $s'_i(En_{\ell}, Ex_{\ell}) > s'_{i-1}(En_{\ell}, Ex_{\ell})$ then
16: \[ s'_i(En_{\ell}, Ex_{\ell}) = \omega \]
17: end if
18: end for
19: end if
20: $i \leftarrow i + 1$
21: end loop
22: $s \leftarrow \text{FLOYD-WARSHALL}(s'_i)$
23: Construct $Gr(A)$ from $s$
24: \[ \text{BELLMAN-FORD}(Gr(A)) \{ \text{Run Bellman-Ford over } Gr(A) \} \]
25: if $Gr(A)$ has a positive cycle then
26: return YES
27: else
28: return NO
29: end if

Algorithm and analysis. Algorithm 3 solves the mean-payoff analysis problem for QICFGs. The computation of the summary function requires $O(|\text{Calls}|)$ computations of the partial summary function $s'_i$, which requires $m$ runs of Bellman-Ford algorithm, each run over a graph of size $|N_{\ell}|$ (hence, each run takes $O(|N_{\ell}|^2)$ time). In addition the computation requires $m$ runs of all pairs maximum weight path (Floyd-Warshall) algorithm. Each run is over a graph of size $O(|N_{\ell}|)$ (hence, each run takes $O(|N_{\ell}|^3)$ time and $O(|N_{\ell}|^2)$ space). Finally we detect positive cycles by running Bellman-Ford algorithm once over the summary graph, which takes $O(|N|^2)$ time and $O(|N|)$ space. Thus we obtain the following result.

**Theorem 30.** (IMPROVED ALGORITHM). Algorithm 3 solves the mean-payoff analysis problem for QICFGs in $O \left( (|\text{Calls}| \cdot (\sum |N_{\ell}|^2) + (\sum |N_{\ell}|^3) + |N|^2 \right)$ time and $O(\sum |N_{\ell}|^2)$ space.

**Remark 9.** Note that in the worst case the running time of Algorithm 3 is cubic and
the space requirement is quadratic.

The next example is an illustration of a run of Algorithm 3.

Example 14. Consider the QICFG that consists of modules f and g (Figures 10.4 and 10.5) and the entry of f is the initial entry of the program. We now describe the run of Algorithm 3 over the QICFG. For simplicity, we denote the graph of f by F and the graph of g by G (and not by $G_1$ and $G_2$). Note that the number of call nodes is 3.

We first compute the summary function $s'_0$ and the first step is to compute $s'_0(f:entry, f:exit) = -35$, and $s'_0(g:entry, g:exit) = -25$.

In order to compute $s'_1(f:entry, f:exit)$ we construct a graph $F^0$ from F by adding a transition from the node f:call g to the node f:ret g with weight $s'_0(g:entry, g:exit)$ and find the maximum weight path from f:entry to f:exit in $F^0$. We get $s'_1(f:entry, f:exit) = -35$. In order to compute $s'_1(g:entry, g:exit)$ we construct a graph $G^0$ from G by adding a transition from g:call g to g:ret g with weight $s'_0(g:entry, g:exit)$ and a transition from g:call f to g:ret f with weight $s'_0(f:entry, f:exit)$ and find the maximum weight path from g:entry to g:exit in $G^0$. We get $s'_1(g:entry, g:exit) = -10$.

Since $s'_1 \neq s'_0$, we continue to compute $s'_2$. We construct $F^1$ and $G^1$ in the same manner as we constructed $F^0$ and $G^0$ (but take the values of $s'_1$ instead of $s'_0$) and get $s'_2(f:entry, f:exit) = -35$, $s'_2(g:entry, g:exit) = 5$. For $i = 3$ we get $s'_3(f:entry, f:exit) = -35$, $s'_3(g:entry, g:exit) = 20$. For $i = 4$, $s'_4(f:entry, f:exit) = -35$, $s'_4(g:entry, g:exit) = 35$. 
For $i = 5$ we get $s_5'(f:\text{entry}, f:\text{exit}) = -20$ and $s_5'(g:\text{entry}, g:\text{exit}) = 50$. Since $i > |\text{Calls}| + 1$ and $s_5'(f:\text{entry}, f:\text{exit}) > s_5'(f:\text{entry}, f:\text{exit})$ and $s_5'(g:\text{entry}, g:\text{exit}) > s_5'(g:\text{entry}, g:\text{exit})$ we assign $s_5'(f:\text{entry}, f:\text{exit}) = \omega$ and $s_5'(g:\text{entry}, g:\text{exit}) = \omega$. In the sixth iteration we get a fix point (that is, $s_6' \equiv s_5'$) and exit the loop block.

From $F^5$ and $G^5$ we compute the summary function $s$. For example $s(g:\text{entry}, g:\text{exit}) = \omega$ and $s(f:\text{entry}, f:\text{exit}) = -30$. Finally, we construct the summary graph (see Figure 10.6) and check whether a positive cycle exists. The cycle $f:\text{entry} \rightarrow f:\text{exit} \rightarrow f:\text{call} \rightarrow g:\text{entry} \rightarrow g:\text{exit} \rightarrow g:\text{call} \rightarrow f:\text{entry}$ contains an $\omega$-edge and thus, it is a positive cycle. Hence algorithm returns YES.

**10.3.2 Efficient algorithm for mean-payoff analysis**

In this section we further improve the algorithm for the mean-payoff analysis problem for $QICFG$s, and the improvement depends on the fact that typically the number of entry, exit, call, and returns nodes is much smaller than the size of the $QICFG$s. Formally, in most typical cases we have $|Ex \cup Retns \cup Calls \cup En| << |N|$. Let $X_\ell = \{Ex_\ell, En_\ell\} \cup \text{Retns}_\ell \cup \text{Calls}_\ell$ and $X = \bigcup_\ell X_\ell$. We present an improvement that enables us to construct the summary function over graphs of size $O(|X_\ell|)$ (instead of graphs of size $O(|N_\ell|)$ of Section 10.3.1), and with at most $O(|\text{Calls}|)$ iterations. Hence, the algorithm in most typical cases will be much faster and require much smaller space.

**Compact representation.** The key idea for the improvement is to represent the modules in compact form. The compact form of a module $A_\ell$, denoted by $\text{Comp}(A_\ell)$, consists of the entry, exit, call, and returns node of $A_\ell$. There is transition between every node in $\text{Comp}(A_\ell)$, and the weight of each transition is the maximum weight path between the nodes with additional stack height 0 (and if there is no such path, then the weight is $-\infty$). Formally, $\text{Comp}(A_\ell) = (V, E)$, where $V = X_\ell$; $E = V \times V$, and $w(v_1, v_2) = s_0(v_1, v_2)$ (where $s_0$ is the bounded height summary function of height 0). If in $\text{Comp}(A_\ell)$ there is a cycle with positive weight that is reachable from the entry node, then we say that $A_\ell$ is a positive mean-payoff witness. The computation of the compact form for a module $A_\ell$ requires $O(|X_\ell| \cdot |N_\ell|^2)$ time and $O(|N_\ell|)$ space (running Bellman-Ford on each $A_\ell$), and thus the compact form for all modules can be computed in $O(\sum |X_\ell| \cdot |N_\ell|^2)$ time and $O(\max |N_\ell|)$ space (note that the space can be reused).

**Witness in summary graph of compact forms.** After constructing the compact forms, we compute a summary function for $\text{Comp}(A_1), \ldots, \text{Comp}(A_m)$, and a corresponding summary graph. We say that there is a path with positive mean-payoff iff there exists a positive cycle in the summary graph or there exists a path to the entry node of a positive mean-payoff witness. The correctness of the algorithm relies on the next lemma.

**Lemma 104.** Let $\mathcal{A} = \langle A_1, \ldots, A_m \rangle$ be a $QICFG$, let $\text{Gr}(\mathcal{A})$ be its summary graph and let $\text{Comp}(\text{Gr}(\mathcal{A}))$ be the summary graph that is formed by $\text{Comp}(A_1), \ldots, \text{Comp}(A_m)$. The following assertions are equivalent:

1. $\text{Gr}(\mathcal{A})$ has a (reachable) positive cycle.
2. Comp(Gr(A)) has a (reachable) positive cycle or a positive mean-payoff witness.

Proof. We first observe that every node in Comp(Gr(A)) exists also in Gr(A) and that the weight of every path in Comp(Gr(A)) has the same weight for the corresponding path in Gr(A) (this can be easily shown by induction over the number of iterations that are needed to obtain a fix point in the bounded height summary function). Hence, if Gr(A) has a positive simple cycle that contains a call node \( c \) for \( c \in \text{Calls} \), then \( c \) is a node also in Comp(Gr(A)) and by the observation above, \( c \) is part of a positive cycle in Comp(Gr(A)). Therefore, Comp(Gr(A)) has a positive cycle. Otherwise, Gr(A) has a positive simple cycle that does not contain a call node. Hence, there is a module \( A_\ell \) with a reachable positive simple cycle that has additional stack height 0. Therefore Comp(Gr(A)) has a positive cycle or a positive mean-payoff witness. This concludes the proof of one direction and the proof for the converse direction is trivial.

The above lemma establishes the correctness of the computation on compact form graphs, and gives us the following result. The following result is obtained from Theorem 30 by replacing \( |N_\ell| \) with \( |X_\ell| \) and \( |N| \) by \( |X| \), and the additional \( \sum |X_\ell| \cdot |N_\ell|^2 \) time and max \( |N_\ell| \) space are required for the compact form computation.

**Theorem 31.** (Efficient Algorithm). The mean-payoff analysis problem for QICFGs can be solved in \( O \left( (|\text{Calls}| \cdot (\sum |X_\ell|^2)) + (\sum |X_\ell|^3) + |X|^2 + \sum |X_\ell| \cdot |N_\ell|^2 \right) \) time and \( O(\sum |X_\ell|^2 + \max |N_\ell|) \) space, where \( X_\ell = \{Ex_\ell, En_\ell\} \cup \text{Retns}_\ell \cup \text{Calls}_\ell \) and \( X = \bigcup_\ell X_\ell \).

**10.3.3 Reduction: Ratio analysis to mean-payoff analysis**

We now establish a linear reduction of the ratio analysis problem to the mean-payoff analysis problem. Given a ICFG \( A \) with labeling of good, bad, and neutral events, a positive integer weight function \( w \), and rational threshold \( \lambda > 0 \), the reduction of the ratio analysis problem to the mean-payoff analysis problem is as follows. We consider a QICFG \( A' \) with weight function \( w_\lambda \) for the mean-payoff objective defined as follows: for a transition \( e \) we have

\[
w_\lambda(e) = \begin{cases} 
  w(e) & \text{if } e \text{ is labelled with good} \\
  -\lambda \cdot w(e) & \text{if } e \text{ is labelled with bad} \\
  0 & \text{otherwise (if } e \text{ is labelled with neutral)}
\end{cases}
\]

The next lemma establishes the correctness of the reduction.

**Lemma 105.** Given a ICFG \( A \) with labeling of good, bad, and neutral events, a positive integer weight function \( w \), and rational threshold \( \lambda > 0 \):

There exists a path \( \pi \) in \( A \) with \( \text{LimRat}(w(\pi)) > \lambda \) iff there exists a path \( \pi \) in \( A' \) with \( \text{LimAvg}(w_\lambda(\pi)) > 0 \).
Proof. Observe that by the definition of \( w_\lambda \) we have that for every \( \epsilon > 0 \) and a finite path \( \pi \):

\[
Rat(w(\pi)) \geq \lambda + \epsilon \text{ iff } Avg(w_\lambda(\pi)) \geq \epsilon.
\]

\( \text{LimAvg implies LimRat.} \) Consider an infinite path \( \pi \). If \( \text{LimAvg}(w_\lambda(\pi)) > 0 \), then by definition there is an \( \epsilon > 0 \) and \( n_0 \in \mathbb{N} \) such that for every \( n \geq n_0 \) we have \( \text{Avg}(w_\lambda(\pi[1, n])) \geq \epsilon \). Hence by the above observation there exist \( \epsilon > 0 \) and \( n_0 \in \mathbb{N} \) such that for every \( n \geq n_0 \) we have \( \text{Rat}(w(\pi[1, n])) \geq \lambda + \epsilon \). Moreover, it follows that in \( \pi \) there are infinitely many edges with positive weights (according to \( w_\lambda \)) and hence \( \pi \) has infinitely many good events. Hence, we get that \( \text{LimRat}(w(\pi)) > \lambda \).

\( \text{LimRat implies LimAvg.} \) The proof for the converse direction is less trivial and relies on properties of QICFGs that we established. Suppose that there is an infinite path \( \pi \) with \( \text{LimRat}(w(\pi)) > \lambda \). We note that every infinite path has infinitely many local minima and let \( c_1, c_2, \ldots \) be an infinite sequence of local minima in \( \pi \). We have the following facts:

1. The segment between every two such configurations \( c_i \) and \( c_j \) for \( i < j \) is a non-decreasing path (since each \( c_i \) is a local minimum).

2. There is a configuration \( c_p \) with a node \( n_p \) such that for every \( \ell \in \mathbb{N} \) there exists a configuration \( c_j \) (for \( p < j \)) such that the segment between \( c_p \) and \( c_j \) is of length greater than \( \ell \) and \( n_p = n_j \), i.e., \( c_p \) and \( c_j \) have the same node (follows from the pigeonhole principle, since the number of local minima is infinite and we have finitely many nodes).

We claim that there exists a non-decreasing finite path \( \pi^* \) that is a segment of \( \pi \), which begins at \( c_p \) and ends at a configuration that has the same node (possibly at different stack height), and we have \( \text{Rat}(w(\pi^*)) > \lambda \). Assume towards the contradiction of the claim that for every configuration \( c_j \) with \( n_p = n_j \), with \( p < j \), we have \( \text{Rat}(w(\pi^*)) \leq \lambda \).

If \( \pi \) has only finitely many good or bad events, then \( \text{LimRat}(w(\pi)) = 0 < \lambda \). Else we consider the following sequence of prefixes of \( \pi \): \( \pi_0 \) is the prefix of \( \pi \) that ends in \( c_p \); and \( \pi_i \) is the segment that starts in \( c_p \) and ends in the \( i \)-th local minimum after \( c_p \) that has the same node \( n_p \). Then we have

\[
\text{Rat}(\pi_0 \cdot \pi_i) \leq \lambda + \frac{w(\pi_0)}{i};
\]

since the length of \( \pi_i \) is at least \( i \). Hence, by definition \( \text{LimRat}(w(\pi)) \leq \lambda \), which establishes the desired contradiction. Thus we have that \( \text{Rat}(w(\pi^*)) > \lambda \) and therefore \( \text{Avg}(w_\lambda(\pi^*)) > 0 \) and the path \( \pi' = \pi_0(\pi^*)^\omega \) is a valid path (since \( \pi^* \) is a non-decreasing path that begins and ends in the same node) with \( \text{Avg}(w_\lambda(\pi')) > 0 \). The desired result follows.

\[\Box\]

Remark 10. Note that in our reduction from ratio analysis to mean-payoff analysis we do not change the QICFG, but only change the weight function. Thus our algo-
rithms from Theorem 30 and Theorem 31 can also solve the ratio analysis problem for QICFGs. Moreover, our proof of Lemma 105 shows that for all paths $\pi$, if we have $\text{LimAvg}(w_\lambda(\pi)) > 0$, then we also have $\text{LimRat}(w(\pi)) > \lambda$, i.e., any witness for the mean-payoff analysis is also a witness for the ratio analysis.

10.4 Experimental Results: Two Case Studies

In this section we present our experimental results on two case studies described in Section 10.2.1 and Section 10.2.2. We run our case studies on several benchmarks in Java, including DaCapo 2009 benchmarks [15], and we use [18, 68] to assist Soot for the construction of the control-flow graphs. First we present some optimizations that proved useful for speed-up in the benchmarks.

10.4.1 Optimization for case studies

We present four optimizations for the case studies: the first two are general, and the last two are specific to our case studies.

**Faster computation of stack height bounded summary function.** We note that if module $A_\ell$ invokes only modules $A_{j_1}, \ldots, A_{j_k}$, and $s'_i(En_{j_h}, Ex_{j_h}) = s'_{i-1}(En_{j_h}, Ex_{j_h})$ for all $h \in \{1, \ldots, k\}$, then $s'_i(En_\ell, Ex_\ell) = s'_{i-1}(En_\ell, Ex_\ell)$. Hence, when computing $s'_i$, we maintain a set $L_i = \{\ell | s'_i(En_\ell, Ex_\ell) > s'_{i-1}(En_\ell, Ex_\ell)\}$, and in the next iteration we run Bellman-Ford algorithm only for the modules that invoke modules from $L_i$.

**Reducing the number of iterations for fix point.** We now present an optimization that allows us to reduce the number of bounded height summary functions from $O(|\text{Calls}|)$ to a practically constant number. We note that the $O(|\text{Calls}|)$ theoretical bound is tight. However, only pathological cases can reach even a fraction of this bound. We note that in typical programs the average nesting of function calls is practically constant (say 10). So if we do not get a fix point after 10 iterations (i.e., $s'_{11} > s'_{10}$), then it is probably because there is a recursive call with positive weight. If this is the case, then if we build the summary graph according to $s_{11}$, we will get a positive cycle in the summary graph, that is, we will get a witness for a path with a positive mean-payoff, and we can stop the computation (since by definition $s \geq s_{11}$, we get that this witness is valid). Hence, our optimized algorithm is to compute the bounded height summary function $s'_i$ and if $s'_i > s'_{i-1}$ and $i = 10, 20, 30, \ldots$, then we construct the summary graph and look for a witness path. If a path is found, then we are done. Otherwise we continue and compute $s'_{i+1}$.

**Removing redundant modules.** Consider an ICFG $A = \langle A_1, \ldots, A_m \rangle$ in which every node is reachable from the program entry (the entry node of the main method). We say that module $A_i$ is **non-redundant** if (i) the module has non-zero weight transitions (good or bad events); or (ii) it invokes a non-redundant module, and is called **redundant** otherwise. Let $A_i$ be a redundant module. For every path $\pi$ that contains a transition to $En_i$ (an invocation of $A_i$), the segment of $\pi$ between that transition and the first transition to $Ex_i$ contains only neutral transitions. Because all nodes of
$\mathcal{A}$ are reachable, we can safely replace each call node that invokes $A_i$ by an internal node that leads to the corresponding return node, and label it as a neutral event. Our optimization then consists of removing redundant modules, as follows:

1. First, we perform a single-source interprocedural reachability from the program entry, which requires linear time ([81]), and discard all non-reachable nodes in all modules.
2. Then, we perform a backwards reachability computation on the call graph of $\mathcal{A}$, starting from the set of all modules that contain non-zero weight transitions. All detected redundant modules are discarded, and calls to them are replaced according to the above description.

Hence, when computing the bounded height summary function, the size of the graph is smaller and the Bellman-Ford algorithm takes less time. Additionally, the number of calls $|\text{Calls}|$ decreases, which reduces the number of iterations required in the main loop of Algorithm 3. In the first case study, typically more than half of the methods are eliminated in this process.

**Incremental computation of summary functions.** We present the final optimization which is relevant for our second case study. Let $\mathcal{A}^1$ be a QICFG and let $\mathcal{A}^2$ be a QICFG that is obtained from $\mathcal{A}^1$ only by increasing some of the transitions weights. Let $s^1$ be the summary function of $\mathcal{A}^1$. Then we can compute the summary function of $\mathcal{A}^2$ by setting $s^2_0 \equiv s^1$ and by computing $s^2_i$ from $s^2_{i+1}$ in the usual way. The correctness is almost trivial. Since the weights of $\mathcal{A}^2$ are at least as the weights of $\mathcal{A}^1$, we get that if we conceptually add a transition $(n_1,n_2)$ with weight $s^1(n_1,n_2)$ for every two nodes (in the same module) in $\mathcal{A}^2$, then the weights of the paths with the maximal weight in $\mathcal{A}^2$ remain the same. By assigning $s^2_0 = s^1$ we only add such conceptual transitions. Hence, the correctness follows. We now describe how this optimization speed up the analysis of the second case study. In the static profiling for function frequencies, we need to build a summary graph for every function $f$, and then run the mean-payoff analysis for every such graph. Given this optimization, we can first compute (only once) a summary graph for the case that all method invocations are bad events. We denote this QICFG by $\mathcal{A}^*$ and the corresponding summary function by $s^*$. Note that in $\mathcal{A}^*$ all weights are negative, and the mean-payoff analysis answer is trivially NO. But still the summary function computation, which computes the quantitative information about the maximum weight context-free paths, provides useful information and saves recomputation. To analysis the frequency of $f$ we assign weights to $\mathcal{A}$ and get $\mathcal{A}^f$. We note that the difference between $\mathcal{A}^f$ and $\mathcal{A}^*$ is only in the weight that is assigned to the invocation of $f$. We then compute the summary function $s^f$ for $\mathcal{A}^f$ by first assigning $s^f_0 = s^*$. In practical cases, programs can have thousands of methods, but only small portion of them will have a path to $f$. So along with the previous optimizations we get that only few Bellman-Ford runs are required to compute $s^f$. Overall, the computation of $s^*$ is expensive, and may take several minutes for a large program, but it is done only once, and for every method the computation of $s^f$ is much faster.
10.4.2 Container analysis

Technical details about experimental results. We discuss a few relevant details about our experiments and results.

- We use the points-to analysis tool of [89]. This tool provides interprocedural on demand analysis for a may-alias relationship of two variables. We say that a variable may point to an allocated container if it may-alias the container, and a variable must point to an allocated container if it may-alias only one allocated container.

- For the underutilized containers the threshold is 1, and for the analysis of overpopulated containers we set a threshold of 0.1 for our experimental results. That is, if the ratio between the number of added elements to the number of lookup operations is more than 10, then the container is overpopulated.

Experimental results. Our experimental results on the benchmarks are reported in Table 10.1. In the table, # M and # CO represent the number of methods and containers that are reachable from the main entry of the program, respectively; # OP and # UC represent the number of overpopulated and underutilized containers discovered by our tool, respectively; and TA(s) and TQ(s) represent the time required for alias analysis and the time required for the quantitative analysis of QICFGs (in seconds), respectively; and the entries of the respective columns represent the time for overpopulated/underutilized container analysis. We now highlight some interesting aspects of our experimental results. First, our approach for container analysis discovers containers that are overpopulated or underutilized, while maintaining soundness. Second, the cases that we identify reveal useful information for optimization, for example, in the first (batik-rasterizer) and the second (batik-svgpp) benchmarks we identify containers that always have a small bounded number of elements.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># M</th>
<th># CO</th>
<th># OP</th>
<th># UC</th>
<th>TA(s)</th>
<th>TQ(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>batik-rasterizer</td>
<td>21433</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>124/125</td>
<td>144/143</td>
</tr>
<tr>
<td>batik-svgpp</td>
<td>7859</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>20/20</td>
<td>14/13</td>
</tr>
<tr>
<td>mrt</td>
<td>9798</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>70/13</td>
<td>41/59</td>
</tr>
<tr>
<td>java-cup</td>
<td>8173</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>19/19</td>
<td>25/22</td>
</tr>
<tr>
<td>xalan</td>
<td>8729</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5/5</td>
<td>41/43</td>
</tr>
<tr>
<td>polyglot</td>
<td>8068</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>0/0</td>
<td>17/17</td>
</tr>
<tr>
<td>antlr</td>
<td>8607</td>
<td>15</td>
<td>5</td>
<td>2</td>
<td>11/12</td>
<td>25/24</td>
</tr>
<tr>
<td>jflex</td>
<td>21852</td>
<td>43</td>
<td>3</td>
<td>6</td>
<td>2473/2614</td>
<td>178/210</td>
</tr>
<tr>
<td>avrora</td>
<td>13331</td>
<td>75</td>
<td>9</td>
<td>9</td>
<td>145/141</td>
<td>111/113</td>
</tr>
<tr>
<td>muffin</td>
<td>22563</td>
<td>50</td>
<td>3</td>
<td>5</td>
<td>2500/157</td>
<td>352/173</td>
</tr>
<tr>
<td>bloat06</td>
<td>10675</td>
<td>211</td>
<td>32</td>
<td>14</td>
<td>399/250</td>
<td>2241/2165</td>
</tr>
<tr>
<td>eclipse06</td>
<td>9335</td>
<td>74</td>
<td>8</td>
<td>4</td>
<td>37/22</td>
<td>222/164</td>
</tr>
<tr>
<td>jython06</td>
<td>12210</td>
<td>66</td>
<td>9</td>
<td>5</td>
<td>154/68</td>
<td>13598/8376</td>
</tr>
</tbody>
</table>

Table 10.1: Experimental results for container usage analysis

Comparison with the work of [108]. While the notion of underutilized and overpopulated containers is the same in our work and in [108], the concrete mathematical definition is different.
Table 10.2: Experimental results for frequency of functions.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># M</th>
<th># I</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>antlr</td>
<td>768</td>
<td>326</td>
<td>1.2</td>
</tr>
<tr>
<td>bloat</td>
<td>2576</td>
<td>676</td>
<td>30.8</td>
</tr>
<tr>
<td>eclipse</td>
<td>1056</td>
<td>215</td>
<td>2.3</td>
</tr>
<tr>
<td>fop</td>
<td>429</td>
<td>47</td>
<td>0.4</td>
</tr>
<tr>
<td>luindex</td>
<td>567</td>
<td>239</td>
<td>0.7</td>
</tr>
<tr>
<td>lusearch</td>
<td>842</td>
<td>237</td>
<td>2.5</td>
</tr>
<tr>
<td>pmd</td>
<td>2547</td>
<td>589</td>
<td>11.5</td>
</tr>
</tbody>
</table>

Figure 10.7: The ROC curves. The left plot shows the results when all methods are analyzed. The right plot shows the results when only the active methods are analyzed.

- (Conceptual difference in modeling). The formal definitions in [108] rely on properties of a specifically constructed inequality graph that in practice gives a good approximation on whether a container is properly used (for details, see Definition 8 in [108]). Our formal definition is conceptually simpler and relies on a very general and flexible mathematical framework, but on the other hand it crucially relies on the accuracy of the constructed QICFG. For example, our technique may not report a container as underutilized even if the witness path for proper utilization is very complex, e.g., has 20 nested function calls, and therefore it is unlikely to be realizable on practice.

- (Advantages and disadvantages). Our approach has several advantages: (1) First, our approach can handle recursion, whereas [108] does not handle recursion. (2) Second, we present a sound and complete approach for ratio analysis of QICFGs, and with a conservative modeling we have a sound analysis approach for detecting containers misuse. (3) Third, our algorithm is polynomial time (once the points-to relation is computed), whereas the algorithmic approach of [108] in the worst case is exponential. (4) Finally, our approach also allows us to handle DELETE operations: in [108] loops with only ADD operations (in ADD vs DELETE) were identified as proper usage, whereas we identify loops where the difference of the ADD and DELETE operations is positive as proper usage (this subsumes the loops of [108] and for example, also loops with two ADD operations and one DELETE operation). Example 12 illustrates the advantages of our approach. One drawback of our approach is that it is conservative: while for un-
Figure 10.8: An example from benchmark batik. The method `run` invokes `cleared` in a loop, and in every invocation, one element of `elementsById` is removed and one element is added. Thus in this loop the total number of elements in elementsById is bounded.

Figure 10.9: An example from benchmark muffin. The method `findRecords` has a recursive call, and method `addCNAME` adds an element to vector `backtrace`. A path with recursion depth $n$ adds $n$ elements to `backtrace`. Hence, `backtrace` may have unbounded number of elements and it is not underutilized.

derutilized containers analysis (Add vs Delete) our approach captures all cases of [108] (as explained above), our approach for overpopulated containers analysis is more conservative (to obtain soundness).

- (Comparison of experimental outcomes). With our approach we were able to fully analyze all the containers in all benchmarks, whereas in [108] the analysis for few benchmarks (e.g., muffin) was not done for all containers since a timeout was reached. Below we present example snippets of code from the benchmarks where our analysis gives different results from the analysis of [108]. The example in Figure 10.8 shows that handling `Delete` operations leads to more refined analysis: in the example, if `Delete` operations are not handled, then the misuse is not detected. The example in Figure 10.9 shows that the proper utilization of containers might depend on the recursive calls. Finally, the example in Figure 10.10 illustrates that the proper use of containers can be outside its allocation site.

10.4.3 Static profiling: frequency of function calls

Experimental results. We examined ten thresholds, namely $1/30, 2/30, 3/30, ..., 10/30$, and for each threshold $\lambda$ we say that a method is
statically hot if it is \( \lambda \)-hot (according to the definition in Subsection 10.2.2). We compared the results to dynamic profiling from the DaCapo benchmarks [15]. In the dynamic profiling we define the top 5% of the most frequently invoked functions as dynamically hot. For example, if a program has 1000 functions, and in the benchmark 500 functions were invoked at least once, then the 25 most frequently invoked functions are dynamically hot. We note that theoretically speaking, the definitions of dynamic and static hotness are incomparable (basically for any \( \lambda \)), but our experimental results show a good correlation between the two notions. To illustrate the correlation we treat our static analysis as a classifier of hot methods, and the specificity and sensitivity of the classifier are controlled by the threshold \( \lambda \). The sensitivity of a classifier is measured by the true positive rate (tpr), which is

\[
\frac{\#\text{dynamically hot methods that are reported as statically hot}}{\#\text{dynamically hot methods}}
\]

The specificity is uniquely characterized by the false positive rate (fpr), namely,

\[
\frac{\#\text{non-dynamically hot methods that are reported as statically hot}}{\#\text{methods}}
\]

For high values of \( \lambda \), the classifier is expected to capture only dynamically hot methods (but it will miss most of the dynamically hot methods), and thus it will have very
high fpr but very low tpr. For very low values of $\lambda$ the classifier will report most of the methods as hot, so most of the hot methods will be reported as hot, and we will have very high tpr but very low fpr. A fundamental metric for classifier evaluation is a receiver operating characteristic (ROC) graph. A ROC graph is a plot with the false positive rate on the X axis and the true positive rate on the Y axis. The point (0,1) is the perfect classifier, and the area beneath an ROC curve can be used as a measure of accuracy of the classifier.

In our experimental evaluation we only considered application functions (and not the library functions), and the results are presented in Table 10.2. In the table, $\# M$ represents the number of application methods (that are reachable from the main entry of the program), $\# I$ represents the number of application methods that were actually invoked in the benchmark, $T$ represents the average running time for the static analysis of a single method (i.e., to check whether a single method is $\lambda$-hot for a fixed $\lambda$) (in seconds). For each $\lambda$ we present the tpr and fpr values of the classifications. We present an evaluation for two cases. In the first case we statically analyze all the methods and calculate the tpr and fpr accordingly. In the second case we consider only the active methods, namely, the methods that were invoked at least once in the program and we remove all the other methods from the program control flow graph. This analysis simulates a typical case where the programmer has a prior knowledge on which methods are definitely not hot and can (manually) instruct the static analysis to ignore them. The ROC curves are presented in Figure 10.7, where the most left points on the graph are for $\lambda = 10/30$ and the fpr and tpr increases as $\lambda$ decreases (until it finally reaches 1/30). In general, for most of the programs the static analysis gives useful and quite accurate information. Specifically, the threshold $\lambda = 7/30$ managed to capture more than half of the hot methods for most benchmarks (namely, other than fop and antlr) and a false positive rate less than 0.3 which means that if a method was statically reported as not hot, then with probability 0.7 it is really not hot. We note that the analysis over fop gives quite poor results because only 10% of the methods were active. However, when we analyzed only the active methods we get better results for fop, see the right hand graph in Figure 10.7. When we only consider the active methods, the threshold $\lambda = 9/30$ manage to capture most of the dynamically hot methods while maintaining a false positive rate less than 0.1 (for most programs).

**Remarks.** We run the experiments over a single thread Intel Pentium 3.80GHz. For Table 10.1 results, for some benchmarks (such as fop, pmd) the alias analysis did not complete. For Table 10.2 results, we only show results for benchmarks for which we managed to obtain the dynamic profiles. Also for a few benchmarks (such as jython) the quantitative analysis took too long for the entire benchmark, and in such cases, our tool could be used to focus on specific functions.
10.5 Related Work

**Interprocedural analysis.** Algorithms that operate on the interprocedural control-flow graphs provide the framework for static analysis of programs, and have numerous applications. Precise interprocedural analysis is crucial for dataflow analysis and has been studied in several works [74, 81]. The study of interprocedural analysis has also been extended to weighted pushdown systems, where the weight domain is a bounded idempotent semiring [22, 83]. Analysis of such weighted pushdown systems has been used in many applications of program analysis [66, 82, 83]. Our work is different because the objectives (mean-payoff and ratio analysis) we consider are very different from reachability and bounded domains. The mean-payoff objective is a function that assigns a real-valued number to every path. In contrast to bounded domain functions, the range of a mean-payoff function is potentially uncountable. We develop novel techniques to extend the summary graph approach for finite-height lattices to solve mean-payoff analysis of QICFGs (which requires computing fix points for infinite-height lattices).

**Mean-payoff analysis.** Mean-payoff objectives are quantitative metrics for performance modeling in many applications and very well-studied in the context of finite-state graphs and games. Finite-state graphs and games with mean-payoff objectives have been studied in [47, 51, 61, 110] for performance modeling, and robust synthesis of reactive systems [16, 17]. Quantitative abstraction-refinement frameworks for finite-state systems with mean-payoff objectives have also been studied in [30]. While the mean-payoff objectives have been considered in depth for finite-state systems, they have not been considered in depth for interprocedural analysis. Pushdown systems with mean-payoff objectives were considered in [38] (and Chapter 7). We significantly improve the complexity of the polynomial-time algorithm for interprocedural mean-payoff analysis that can be obtained by a reduction to the results of Chapter 7.

**Detecting inefficiently-used containers.** Bloat detection and detecting inefficiently used containers have been identified in many previous works as a major reason for program inefficiency. Dynamic approaches for the problem were studied in many works such as [46, 72, 73, 76, 87, 88, 107]. A static approach to analyze the problem was first considered in [108], which is the most closely related work to our case study. The work of [108] provides an excellent exposition of the problem with several practical motivations. It also describes the clear advantages of the static analysis tools, and identifies that soundness in detecting inefficiently used containers (with no or low false positive rates) is a very important feature. Our approach for the problem is significantly different from the approach of [108]. A big part of the contribution of [108] is an automated annotation for the functionality of the containers operation. The main algorithmic approach of [108] is to use CFL-reachability (context-free reachability) to identify nesting loop depths and then use this information for detecting misuse of containers. Our algorithmic approach is very different: we use a quantitative analysis
approach, i.e., ratio analysis of QICFGs to model the problem.

**Static profiling of programs.** Static and dynamic profiling of programs is in the heart of program optimization. Static profiling are typically used in branch predictions where the goal is to assign probabilities to branches, and typically require some prior knowledge on the probability of inputs. Static profiling of programs for branch predictions has been considered in [11, 55, 100, 105]. Dynamic profiling has also been used in many applications related to performance optimizations, see [103] for a collection of dynamic profiling tools. Two main drawbacks of dynamic profiling are that they require inputs, and they cannot be used for compiler optimizations. We use static profiling to determine if a function is invoked frequently along some run of the program, and do not require any prior knowledge on inputs. The techniques used in [11, 100, 105] involves solving linear equations with sparse matrix solvers, whereas our solution method is different (by quantitative analysis of QICFGs).

### 10.6 Conclusion

In this work we considered the quantitative (ratio and mean-payoff) analysis for interprocedural programs. We demonstrated how interprocedural quantitative analysis can aid to automatically reason about properties of programs and potential program optimizations. We significantly improved the theoretical known upper-bound for the polynomial-time solution, and presented several practical optimizations that proved to be useful in real programs. We have implemented the algorithm in Java, and showed that it scales to DaCapo benchmarks of real-world programs. This shows that interprocedural quantitative analysis is feasible and useful. Some possible directions of future works are as follows: (1) extend our framework with multiple quantitative objectives and study their applications; and (2) extend [30] to have an abstraction-refinement framework for quantitative interprocedural analysis.
Chapter 11

Summary and Further Work

In this thesis, we studied multidimensional mean-payoff games over finite graphs and mean-payoff games over infinite-state pushdown graphs. For finite graphs, we showed a pseudo-polynomial algorithm for two-player games with conjunctive objectives and for arbitrary conditions we gave tight complexity bounds both for the model-checking problem and for two-player games where either both, one or none of the players are restricted to play only with finite-memory. For pushdown graphs we showed that two-player games are decidable only when the protagonist is restricted to play with modular strategies and we presented a polynomial algorithm for one-player games with global strategies for conjunctive multidimensional (and for single-dimensional) objectives. In addition we presented two applications for our research: the first application is a theoretical contribution for the study of the complexity of equilibria in multi-player games, and the second application is a tool that automatically reason about quantitative properties of Java programs.

The next items are possible directions for further work:

- In Chapter 3 we presented a polynomial algorithm for mean-payoff games with conjunctive multidimensional condition over finite graphs with bounded (fixed) number of dimensions and weights. The complexity of games with a fixed number of dimensions and with arbitrary weight was not settled. A trivial coNP upper bound follows directly from [99], where we proved coNP upper bound for games with arbitrary weights and number of dimensions. It is an open question whether the problem is also in NP and whether there is a reduction to a single-dimensional mean-payoff game (the reduction in Chapter 3 is polynomial only for bounded weights).

- In Chapter 6 we showed that two-player games with arbitrary strategies over finite graph with arbitrary multidimensional objectives are undecidable by a reduction from the problem of determining whether player 2 is the winner to the halting problem. Hence, the problem is RE hard and thus not in coRE (where RE is the class of recursively enumerable problems). It is an interesting problem to determine whether the problem is in RE. Another interesting open question is the computability of \( \epsilon \)-optimal strategies when the objective is a mean-payoff
expression. We recall that in Chapter 5 we proved the computability of $\epsilon$-optimal finite-memory strategies for mean-payoff expressions.

- In Chapter 7, we proved that two-player (single-dimensional) mean-payoff games over pushdown graphs are undecidable. The decidability of mean-payoff games over one-counter graphs remains open. (One-counter graphs are special cases of pushdown graphs with a single stack alphabet.) Recently, Abdulla et al., [1] proved that for energy objectives, where the goal of the protagonist is to assure non-negative sum of weights along the entire (infinite) play, determining who is the winner is undecidable when playing over pushdown graphs, but decidable when playing over one-counter graphs.

- In Chapter 10 we implemented a tool for quantitative interprocedural analysis of Java programs. A future work is to implement the algorithm for conjunctive multidimensional one-player pushdown game from Chapter 7 and to get a tool that automatically reasons about multiple quantitative properties of a program.
Bibliography


[57] Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, 22(3):384–406, 1981. 3.2


[75] Katta G Murty. Linear programming, volume 57. Wiley New York, 1983. 4.2.5


[80] Amir Pnueli and Zohar Manna. The temporal logic of reactive and concurrent systems, 1992. 4


[84] Aaron Roth. Personal website, 2012. 1.2.3.1


[108] Guoqing Xu and Atanas Rountev. Detecting inefficiently-used containers to avoid bloat. ACM Sigplan Notices, 45(6):160–173, 2010. 1a, 3a, 10.2.1, 10.2.1, 12, 7, 10.4.2, 10.10, 10.5

משחקי תשלומים ממוצע על גבי גרסה מכ-435
مصקריואוטומיות מסכימות עם יישומי בביומד
 עבודת蝉תח על תכונות: סיבוכיות, כרירות
ואלגוריתמים

innacle לשכנך החוז_rankו דוקטור לפילוסופיה
מאות
ירון וולר

עבודה בו נושאת בחדרות
פרופ' אלכסנדר רביןוביץ'
הווש לשכנך של אוניברסיטת תל-אביב
וליל 2015
The abstract:

The mean-payoff games played on graphs have gained importance in verification and synthesis of automata. In recent years, there has been much research to include Boolean verification techniques and enable the players to infer quantitative properties of programs. 

Typically, quantitative properties are modeled by a weight function that assigns a weight to each program statement, and the common metric assigns the average payoff (mean-payoff) of the weights that appear during the run.

The synthesis problem is modeled by a game with two players and an infinite number of rounds, and a winning strategy for the first player constitutes a correct implementation of the program. The verification problem is modeled by a game with one player, and a winning strategy for that player is a sign of the existence of an unwanted path in the program.

Mean-payoff games are played on weighted graphs. The vertices of the graph are divided between the two players, and the player whose vertex is current decides the next move in the game. The goal of the first player is to ensure an average impartial weight during the game, and the goal of the second player is to achieve an average weight.

These games were first proposed for solving and optimizing problems in economics, but in recent years, they have been extensively studied in the area of verification and quantitative synthesis of programs.

The purpose of this work is to extend the classical research on mean-payoff games in two directions:

The first direction is the study of mean-payoff games played on graphs with multi-dimensional weight functions. A multi-dimensional weight function can model multiple properties of the program (or conflicting properties).
For example, the energy consumption and response time of the program.

In mean-payoff games with multi-dimensional weight functions, the goal of the first player is to provide a Boolean formula for the average payoff of the different dimensions.
For example, he must achieve a greater than 1 in the first dimension and in addition, achieve a minimum of 7 in the second dimension or a minimum of 5 in the third dimension.

The second direction is games played on infinite graphs generated by automata (pushdown automata).

These graphs allow modeling of control flow of recursive programs. 
תרומותינו העיקריות בעבודה זאת הן:

1. משחיקוני עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים bénéficie על סימול פנים וממוצע איזו סכרים אינדיבידואליים.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

4. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

5. משחיקונים על מהפכניים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים על גב ענייןuggage פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

4. משחיקונים על גב ענייןuggage פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

5. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים על גב ענייןuggage פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בעבודה זאת הן:

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בע命中ן

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי.

תרומותינו העיקריות בע命中ן

1. משחיקונים עניין Về פונקציות משקל רב ממדיות: אנוקגפונות והנשים ניצוץ במתח (conjunctive) ובמסור השחקן.

2. משחיקונים.FontStyle על שחקנים איזו ניצוץ בכלי בטא השחקן והנשים מוגבל.

3. משחיקונים FontStyle על שחקנים איזו ניצוץ בכלי.
 אנו משתמשים בטכניקות לפתרוןمشחכים רב מימדעם מחשב אחד על מנת לahoma חסמי סיבוכיות הדוקים
לבעיית מציאת שווי המשקל במשחקים רב
משתתפים עם אינסוף סיבובים בהם פונקצית התשלום מיוצגת על ידי מסריץ.

 אנו מישמשים את התוצאות עבור משחקים על גבי אוטומט מחסנית ובונים כלי
המנחת במאפים ו.AppendTextים תכונות כמותיות של תוכניות

(ב) הנחת במאפים וארטומט תכונות כמותיות של תוכניות
תקציר

בפרק זה נציג את הרקע של התזה ואת תרומותיה העיקריות. בתת הפרק הבא נסקור בקצרה את בעיית האימות האוטומטי ומשחקים על גבי גרפים. בתת הפרק העוקב נציג את תרומתנו העיקריות ומשווה אותן ל obsłוע�ות קודמות בנושא. לבסוף, בתת הפרק ההווה נציג את רשימת הפרסומים עליהם מבוסס מחקר זה.

הערות ביבליוגרפיה: רוב התוצאות המוצגות בעבודה זו פורסמו בעבר בכנסים. בתת הפרק האחרון אנו נותנים רשימה מלאה של הפרסומים על של הפרסומים על של הפרסומים על של הפרסום זה.

רקע

בתת פרק זה אנו מציגים את הרקע ההכרחי עבור משימות האימות והסינתזה האוטומטיות, ואת הקשר של השם מחוסק ב Bravo אוטומט ופירמידה הפורמלית והסינונימיית האוטומטיות, ואת האתחולית האוטומטיות והסינונימיית הכמותיות ואת המיתר הקלטי עצור של השפק של מהקל.

אימות, סינתזה ומשתנה על גבי גרפים: אימות אוטומטי של תוכנית שהיינו הפורמלית והסינונימיית הכמותיות ואותו גרף. פורמט שבר בבלב של התוכנית והסינונימיית את הרקע של אימות והסינונימיית במישורי בנימין על ידי התוכנית והסינונימיית בדווכת,agoon, 2019, 60]. מישמי הקש המיתר方は (לסיפת) אוによって המיתר הקלטי את האיתון הפרמאלית.

תחום הiatric והפורמלית, מישמי האיתון מתוכננת על ידי מודל המיתר הפורמלית, Лור פורמלית בשתי קבלי, כשריעה שאיינה מתכנתת על ידי האיתון מייצג ריצה ולא ריצה של התוכנית (ולכד התוכנית את נוכחה). מישמי הסינונيميית המיתר קלטי כלשון אינספור (国际在线 סיסבום) בסין שᾏה על נור. במשתנה החלל, זצמי הכף מחלכים בינ השחקה הראשה וה⎥ון. מישמי מהחלי במרומת התחהל,‼️ נור סיסבום במשח, השחקן אדר השיקᑐ. 
The current paper presents the game at the intersection since it is played endlessly, and the outcome of the game is an infinite path in the graph known as the game. The first player wins the game if the path created (represents a run in the program) satisfies the feature. Otherwise, the second player wins.

A strategy for a player is a recipe showing how to decide the next move in the game. A strategy is called a winning strategy if it guarantees a win against any opposing strategy by the opponent player.

The collection of all the strategies of the second player represents the collection of all possible runs of the program.

Therefore, a winning strategy for the first player represents a correct use of the program. Consequently, the synthesis problem is equivalent to the computer algorithm strategy of the first player.

In the context of games on graphs, the verification problem transforms to a game with a one player in which he owns all the vertices in the graph and therefore he decides all the moves in the game. A winning strategy for the first player is an infinite path in the graph representing a run in the program.

Therefore, a strategy violating the feature is evidence of the existence of a possible run in the program that does not satisfy the feature.

Classical research on verification and synthesis of automata dealt with Boolean properties of the program, such as living (the program is always in a good state) and safety (the program will never enter a bad state).

Two notable points in this field are Buchberger [79], [107], and Parikh [79], who introduced the use of temporal logic (temporal logic) as a central tool for solving verification problems, and Moshe Fischer-Booker [79] who proved that one can solve the synthesis problem for a Programmable Automaton with an infinite feature.

Verification and synthesis quantitatively: In recent years, many studies aimed to extend the formal methods classically used for quantification of features and systems, see [79], [120], [15], [33], [49]. Unlike the classical methods in which only whether the feature is satisfied or violated, the quantified feature aims to define to what extent the program satisfies or violates the feature.

A feature quantified assigns a numerical value to every run in the program, (not just a Boolean value).

Quantitative verification measures how much the program satisfies the feature, i.e., calculates the minimal value for an optimal run of the program.

Synthesis aims to find the optimal implementation of the program in terms of features. A quantified feature is typically modeled by a finite automaton [34] and the studied feature is the average reward over all rewards.

Games of average rewards on graphs: Games of average rewards are played on a weighted and infinite graph. Weight (represents the reward) can be assigned to each edge in the graph.

The objective of the first player is to maximize the reward.
嵂ואית התזה ואיבודות קדומות

בתח פק זה是我们 מפרטים את התרומתנו העיקרית וסיקורינו עבדות קדומותmodelo של התזה. החלה możemy内科 במאמר, הנתונים שהに乗וckiיה התרומתנו, לא כמו גם מספריים שונים של התרומתנו. חלקים של התהנה ערכים ותהליך חילקי להתקדם במאמר.

מרחביים עלייה תלה תיוריות בטיחון ו להשפיע במאמר לקדם את התהנה וכמה מאמציים שונים של התרומתנו. באחרון למקרים קדומים של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות בטיחון ו להשפיע בשיטה אחרונה של לוחות לטיחון
בעות של סינתזה מחרוזתית ייחודית לשפה של השפה המופיעה על פונקציה משקל בר

ממדית.

משחקים על פי אוטומטים מחסנית, ולא מספר האחד, משחקים על גבי מכונים רקורסיביים ממדים

ואז עבור שעון זווית האמת, סינトー להנקת אנושות על תחתון [8,9,10,102].

ולפיים, משחקים והשלמה מעורר על גבי אוטומטים מחסנית, משחקים על גבי גרף סופי פונקציה

משקל בר ממדית הינו בעית ביסוס ניבייה בתchosen החינוך וה->___ה של לוטמרכזים

בנוסף, בעובדה, אנו מחזיאים שים י發布ים של מחקר בנושאים אלו. הנויים האידאטיוז והא

בנתה תורת המשטרונים הקלאסיים: ואנו מחזיאים שיתוף לפתרון משחק שלשלמה מעורר על

շחק אחד משלים בר ממדיה מוכרים חסם סיגנלים שלטו מודרני צוואת עליון משקל בשישה

בר משלים עם מספר סיבובים חסם ופונקציה המשטרון מופיעה עם עליון מטריציה (ולא עם

גרף). הנויים נשיא היא בתchosen החינוך החשובים של תחתון: אנו מחזיאים את האלגוריתם

(inter-procedural) הפרוסנס עבר משחקים על גבי אוטומטים מחסנית עבזת גיתות בכר-

שמאות עבזת משחקים על גבי אוטומטים מחסנית עבזת גיתות בכר-

שהתונים כמותית על תחתון.

ואנ מתוויי ביכר פריטים בא 한יה של תחתון ובשלאות חתי הפרקים הקטנים הבאה. בת

הפרק ה肫ן האידאטיוז והאטניים את הת合わונות עבזת משחקים על גבי גרף סופי. בוח הפרק

ה zwłאיי והאטניים את הת合わונות עבזת משחקים המשקפים העבזת גיתות מחסנית. בוח הפרק

ה zwłאיי יישואים עבזת יישואים עבזת תחתון.

משחק שלשלמה מעורר על פונקציה משקל בר ממדית על יבר גפרים סופיים

בנמשחק שלשלמה מעורר על פונקציה משקל בר ממדית הת훈אות של משחקן על גבי גרף k-

והא קוקו של k תשלומיים מעורר. התשלומיים מעורר על מימד 1 קוקו על סלק המשקל

הcomedיקيمي של. מאמור החשק האידאטיוז היה למספק תסנט开元棋牌ים מעיל רפי התשלום

המודיעות, ואו תלנו, להבアイ למוקסימום את האור של פונקציה כמותית המודיעה מעיל רפי

התשלומיים.
The Boolean condition over the average payment vector (\(p_1, p_2, \ldots, p_k\)) is defined using the vector of thresholds (\(t_1, t_2, \ldots, t_k\)) and the disjunctive normal form:

\[\phi(p_1 > t_1, p_1 \geq t_1, \ldots, p_k > t_k, p_k \geq t_k).\]

For example, \((p_1 \geq 2) \land (p_2 > 7) \lor (p_1 < 9),\) where \(p_1 < 9\) is a shortcut for \(\neg(p_1 \geq 9).\)

A special case is the Boolean condition that each average payment \((\text{in all dimensions})\) should be non-negative.

An expression is a function that returns a numerical value for each vector of average payments.

The counterparts to Boolean operators \(\land, \lor, \neg\) are algebraic operations \(\max, \min\) and complements \((2, 3).\) In addition, the operation of addition is also a meaningful algebraic operation.

We define a recursive class of average payment expressions as follows:

- The average payment of a specific dimension is an expression (e.g., \(p_7\)).
- If \(E_1\) and \(E_2\) are expressions, then \(\max(E_1, E_2), \min(E_1, E_2), E_1 + E_2\) and \(\neg E_1\) are also expressions.

For example, \(\max(p_1, p_2 + \min(p_3, p_4))\) is an expression.

We can use these formulas to check whether a player can win in a one-player game, and only if they can guarantee a non-negative value for \(E\).

Previous work on the verification problem (games with one player) for multi-dimensional weighted functions was first studied by Adler et al.\(^7\) who researched the verification problem for Boolean conditions over average payments.

Boker et al.\(^1\) and Tomita et al.\(^9\) studied extensions that allow the integration of Boolean conditions on an average payment and Boolean conditions on a regular feature.

Zietz et al.\(^3\) presented and analyzed this class of expressions.

Tests were shown to be coNP-complete for every question.

Games with two players and a general Boolean condition (or an average payment condition) are studied in this paper.

The main result is that the verification question whether the first player can win is coNP-complete.
1. אנו מציגים אלגוריתם פולינומי לゴール שני של שחקן עם תנאים שלבים מרוכבים וממשחים על רדיקציה של השחקון שרגלי."
2. אנו מציגים אלגוריתם המשמש בשחמטור ירדניות פולינומי城堡 (PSPACE)אירוע בעיית האימנה (משחקים בין שחקן אחד) של במור 쉴 צלול מצטמצם. אנו מראים כי Spells נ práctica הולכת והולכת ב PSPACE שולם הקולנוע. התשובה התשובה של הפתרונות של השחקון הוא פופולארי.
3. אנו מציגים אלגוריתם שמשתמש בполь אקראי צג ופונקציה של מספר בצורת פולינומי城堡 של הריבוע של השחקון-handed. אנו מראים כי סיבוכיות האלגוריתם היא פופולארי (כלומר, הבעיה היא PSPACE שלמה).
4. אנו מראים למקרה הזה של השחקון שהוא בניצחון ב jogo פולינומי城堡 של שחקון ב PEXPTIME.

 élevé הקדוק, אנו מראים כי בעיה זו שקולה לבעיה השישית של הילברט עבור מספרים רציונליים שאפשרונה לא ידוע עד עצם היום הזה.

משחקון של işleומע מנותני משחיק באופן רכיבי על רב פפרים

המתحك הקלסתי העוסק בשחקון הוא על נטרס ספיני והגי נטרס בוליאניות והורחק בשני

הὋיא כך שחקון הוא משחקון על נטרס ספיני והגי נטרס בוליאניות, היזגון

הנתחים של השחקון, הוא משחקון על נטרס ספיני והגי נטרס בוליאניות, היזגון

השישה של השחקון, הוא משחקון על נטרס ספיני והגי נטרס בוליאניות, היזגון

הממד של השחקון בין נטרשים זה משחקון בולייגה על נטרס של אוטומטים משחיקים

המידה של שחקון כזה, משחקון בין נטרס של אוטומטים משחיקים על נטרס זה של

קוליאריונים מחקור ב [8,9,101.102]. מארקנט אמצות השחקון בין נטרס זה או אובפטיק הקשה

הממד המוצע. לפיכך משחקון של işleומע מנותני על רב פפרים משחיק באופן רכיבי שילוב שילוב.
בעבודה זו אנו חוקרים לראשונה משחקים ממוצע על גבי גרף של אוטומט מחסנית. במשחキים למעל אוטומטים מחסנית ישנן שני סוגי של אסטרטגיות. הסוג הראשה היה אסטרטגיות גלובליות בה החל מחזור בחר במיקום בגלובלי עם סכום כל הצעדים שתחבבנו ב昄 של hakkı. הסוג השני היה אסטרטגיות מודולריות. אסטרטגיות אלו מצויות על ההחלקה על旗帜 ההמאית הקדימה.

העדויות של המחקרים באנפוגור.

משחיקי על גבי אוטומטים מחסנית עם תנאים בינאריים ניגודים ב[20,21]. ב[22] הוכחה כי משחיק ממוצע עם תארך בינארי הוא EXPTIME ניתן (reachability) EXPTIME הם与时 באנפוגור מודולריים,aupt החכם שוחה בו EXPTIME. ליצין משחיק עם תארך בינארי הוא.

משחיק אסטרטגיות המודולריות חוג ב[8,9]. הוכה כי בעיית מציאת אסטרטגיות מודולריות

מנחה בכדי את ניצחון או ניצחון הים שלמה.

הערות

בעבודה זו, ייחד עם שלחנדו צ'טריזי, אנו מוכחים התחומים של אוטומטים מחסנית עם תארך בינארי, אסטרטגיות מודולריות, את הסכמת תחרות בין אוטומטים מחסנים עם鲑 עב...

העיקויות והכללים:

משחיק אסטרטגיות גלובליות: אנו מציאים שלמשחיק בום עם שחקן אחד בברד

产量ס של תחרות בכר מפלטוניאלי בעד בשובשהמשחיק עם鲑 שחקנים, מיציאת המגיעה

产量ס בשתי בריכת יער. בנסף את השחקן עם המשחיק עם鲑 שחקן אחד והזזתי יער רב

מפרים מתחבר מומלים עם מלק.apply הכזב פלטוניאלי.

משחיק אסטרטגיות מודולריות: אנו מציאים שלמשחיק בום השחקן

המשחיק יובן אסטרטגיות מודולריות כדי מיציאת השחקן המגיעה לא עב

שלמה במסע שחקנים עם鲑 שחקן אחד ומשחיק עם鲑 שחקנים. בנסף את

产量ס עם鲑 שחקנים עם鲑 יער רב מפרים מתחבר מומלים כדי יער עב

מפרים תחרות עם鲑 שחקנים.
ישומי

אנו מציגים שני יישומים של מחקרנו. היישום הראשון הוא יישום תיאורטי בתורת המשחקים הקלאסית, בה תשלומי השחקנים מיוצגים באמצעות מטריצה ולא באמצעות גרף. היישום השני

היא לשניים סטטי של תוכנות מחשב של תוכנות מחשב.

1) תشور שון משקל במשתוקים אינסופיים המשוקות בחזרות: וחקירים משוקקים

2) ניתוח סטטי של תוכנות מחשב: ייחוד סטטי של תוכנות מחשב

3) ניתוח סטטי של תוכנות: ייחוד סטטי של תוכנות

4) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

5) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

6) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

7) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

8) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

9) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

10) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

11) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

12) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

13) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

14) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

15) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

16) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

17) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

18) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

19) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

20) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

21) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

22) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

23) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

24) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

25) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

26) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

27) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

28) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

29) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

30) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

31) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

32) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

33) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

34) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

35) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

36) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

37) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

38) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

39) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

40) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

41) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

42) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

43) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

44) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

45) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

46) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

47) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

48) ניתוח סטטי של תוכנות מחשב: ת络ית סטטי של תוכנות מחשב

49) ניתוח סטטי של תוכנות מחשב: ת絡ית סטטי של תוכנות מחשב

50) ניתוח סטטי של תוכנותמחשב: ת络ית סטטי של תוכנות מחשב
The negative events. We present a polynomial algorithm for solving this problem, written in JAVA and we present test cases that show that the algorithm is useful and runs in reasonable time.

Publications

This work is based on a research that was published in conferences.

Below is a list, in chronological order, of the publications upon which the thesis is based.

1. Krishnendu Chatterjee, Yaron Velner: Mean-Payoff Pushdown Games. LICS 2012.
7. Yaron Velner: Robust Multidimensional Mean-Payoff Games are Undecidable. FOSSACS 2015.