RICE UNIVERSITY

Automata-Based Quantitative Verification

By

Suguman Bansal

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Doctor of Philosophy

APPROVED, THESIS COMMITTEE

M. Vardi
Moshe Vardi (May 11, 2020)

Moshe Vardi

Karen Ostrum George Distinguished Service Professor in Computational Engineering, Rice University

Rajeev Alur
Rajeev Alur (May 11, 2020)

Rajeev Alur

Zisman Family Professor of Computer and Information Science, University of Pennsylvania

Konstantinos Mamouras
Konstantinos Mamouras (May 11, 2020)

Konstantinos Mamouras

Assistant Professor of Computer Science, Rice University

Peter Varman

Peter Varman

Professor of Electrical and Computer Engineering, Rice University

HOUSTON, TEXAS

May 2020

Dedicated to my family,

My mother, Anita, who now resides among the stars

My father, Rajeev

My brother, Sanchit

ABSTRACT

Automata-Based Quantitative Reasoning

by

Suguman Bansal

The analysis of *quantitative properties* of computing systems, or *quantitative analysis* in short, is an emerging area in automated formal analysis. Such properties address aspects such as costs and rewards, quality measures, resource consumption, distance metrics, and the like. So far, several applications of quantitative analysis have been identified, including formal guarantees for reinforcement learning, planning under resource constraints, and verification of (multi-agent) on-line economic protocols.

Existing solution approaches for problems in quantitative analysis suffer from two challenges that adversely impact the theoretical understanding of quantitative analysis, and large-scale applicability due to limitations on scalability. These are the *lack of generalizability*, and *separation-of-techniques*. Lack of generalizability refers to the issue that solution approaches are often specialized to the underlying *cost model* that evaluates the quantitative property. Different cost models deploy such disparate algorithms that there is no transfer of knowledge from one cost model to another. Separation-of-techniques refers to the inherent dichotomy in solving problems in quantitative analysis. Most algorithms comprise of two phases: A *structural phase*, which reasons about the structure of the quantitative system(s) using techniques from automata or graphs; and a *numerical phase*, which reasons about the

quantitative dimension/cost model using numerical methods. The techniques used in both phases are so unlike each other that they are difficult to combine, forcing the phases to be performed sequentially, thereby impacting scalability.

This thesis contributes towards a novel framework that addresses these challenges. The introduced framework, called *comparator automata* or *comparators* in short, builds on automata-theoretic foundations to generalize across a variety of cost models. The crux of comparators is that they enable automata-based methods in the numerical phase, hence eradicating the dependence on numerical methods. In doing so, comparators are able to integrate the structural and numerical phases. On the theoretical front, we demonstrate that comparator-based solutions have the advantage of generalizable results, and yield complexity-theoretic improvements over a range of problems in quantitative analysis. On the practical front, we demonstrate through empirical analysis that comparator-based solutions render more efficient, scalable, and robust performance, and hold the ability to integrate quantitative with qualitative objectives.

List of Publications

This thesis is based on the following publications. Publications 1-2 are yet to appear in an official proceedings at the time of thesis submission.

1. **On the analysis of quantitative games**
   Suguman Bansal, Krishnendu Chatterjee, and Moshe Y. Vardi

2. **Anytime discounted-sum inclusion**
   Suguman Bansal and Moshe Y. Vardi

3. **Safety and co-safety comparator automata for discounted-sum inclusion**
   Suguman Bansal and Moshe Y. Vardi
   In Proceedings of International Conference on Computer-Aided Verification (CAV) 2019

4. **Automata vs linear-programming discounted-sum inclusion**
   Suguman Bansal, Swarat Chaudhuri, and Moshe Y. Vardi
   In Proceedings of International Conference on Computer-Aided Verification (CAV) 2018

5. **Comparator automata in quantitative verification**
   Suguman Bansal, Swarat Chaudhuri, and Moshe Y. Vardi
   In Proceedings of International Conference on Foundations of Software Science and Computation Structures (FoSSaCS) 2018
   (**Extended version** with additional results on Arxiv)

Acknowledgements

Foremost, I would like to thank my advisor and mentor Moshe Vardi for his unwavering support, constant guidance, and encouragement to pursue my ideas. He took me under his wings during the most difficult times I have faced, both academic and personal. Needless to say, there has been no looking back since. Thank you for giving me the freedom for ample exploration while also nudging me towards the right path. Most importantly, thank you for believing in me. I hope to be the advisor you have been to me to another dreamer someday.

I am fortunate to have worked with an excellent thesis committee: Rajeev Alur, Konstantinos Mamouras, and Peter Varman. Their suggestions, feedback, and thorough evaluations of this thesis have led to numerous improvements and pursuits for future work. Thanks go to Krishnendu Chatterjee and Swarat Chaudhuri, collaborators on parts of this thesis. Their feedback on my doctoral research from the very beginning has been crucial in shaping the course.

I am grateful to have found mentors in Swarat and Kedar Namjoshi. Right from building my foundations in Computer Science, Swarat has helped me navigate through the bigger career decisions. Incidentally, it was upon his advice that I took up the internship offer from Bell Labs to work with Kedar, which has turned out to be one of my best collaborative experiences. The daily, intense brainstorming sessions with Kedar were tiring yet immensely fulfilling. That said, a respite from the sweltering Houston weather takes the cherry on the cake for both of my summers at Bell Labs.

During the Ph.D., I had the opportunity to collaborate with several brilliant researchers from around the globe: Rajeev Alur, Shaull Almagor, Swarat Chaudhuri, Krishnendu Chatterjee, Dror Fried, Yong Li, Kuldeep Meel, Kedar Namjoshi, Yaniv

Sa'ar, Lucas Tabajara, Moshe Vardi, Andrew Wells. I am, perhaps, most partial to collaborations with my peers at Rice - Yong Li and Lucas Tabajara - where discussions would begin in our offices but end in Valhalla (*Viva Valhalla!*).

The fact that there is a large overlap between my friends and (extended) research group LAPIS is a testament to how instrumental they have been in this journey. Dror Fried has been so much more than a friend to me. I am eternally grateful to his wife Sagit, him, and their three little kids (not so little anymore) for giving me an abode in their hearts. In a discipline that suffers from a dearth of women, I have been very lucky to have Afsaneh Rahbar and Shufang Zhu by my side. These women have truly made my successes and failures their own, as we continue to inspire each other with our *"We can do it"* chant. Because Aditya and I share a history of borrowing each other's sentences, I'll paraphrase him here - Thank you to Yong Li and Aditya Shrotri for keeping me company during the late nights and weekends in the office, the car rides home, coffee breaks, lunches, and dinners. I am humbled to share a special siblings-like bond with Vu Phan. I have relied on Jeffery Dudek, Antonio Di Stasio, my officemate Lucas Tabajara, Kevin Smith, Abhinav Verma, and Zhiwei Zhang for cerebral discussions and simple reasons for laughter.

My life outside the colorful walls of Duncan Hall would have been so lack-luster had it not been for the friendships of Priyadarsini (Priya) Dasari, Vaideesh Logan, Rakesh Malladi, Sushma Sri Pamulapati, and the *folks@IMT*. In addition to keeping our apartment in pristine condition, my flatmate Priya has made me a better cook, and counseled me through personal and work-related turmoils; all of this over a steaming cup of *chai*. The *folks@IMT* have been a source of strength in the times of social-distancing/isolation during the ongoing Covid-19 pandemic. In these uncertain times, their small gestures have gone a long way to ease the anxiety around the global upheaval. Thank you all for making Houston my home.

My friends from undergraduate and school have been available in moments of despair and self-doubt. I cannot thank Saheli, Siddharth, Siddhesh, and Visu enough

for having been just a phone call away for almost a decade.

Of all the things I had imagined graduate school would entail, meeting the love of my life was not on the agenda. Yet, here we are. Whether we are ten centimeters or ten thousand miles apart, Kuldeep can always make me smile. Thank you, *ghano saaro*, for brightening my day, every day.

Finally but most importantly, I owe my deepest gratitude to my parents, Anita Bansal and Rajeev Kumar, and brother Sanchit Bansal. Right since I can remember, they filled me with curiosity, creativity, and imagination, and have shunned societal norms so that I could run, stumble, and chase after my dreams. Their love and support have been unconditional. It is to them that I dedicate this thesis.

Yet there is a pit in my stomach. We lost my mother, Anita, four years ago. She was my loudest champion and strictest critic. Never had I imagined crossing this milestone without her cheering from the stands. But I am not upset, because I know that wherever she is, she is very proud of her *guriya*. *We miss you every day, Mummy!*

Contents

Illustrations

List of Algorithms

Chapter 1

Introduction

Formal methods offers rigorous mathematical guarantees about properties of systems. Two cornerstones of formal methods are (a) *Verification*: Given a system and a property, does the system satisfy the property, and (b) *Synthesis*: Given a property, does there exist a system that satisfies it? These strong guarantees come at the price of high computational complexity and practical intractability. Yet, diligent efforts of the last few decades have resulted in the emergence of formal methods as an integral asset in the development of modern computing systems. These systems range from hardware and software, to security and cryptographic protocols, to safe autonomy. In most of these use-cases, systems are evaluated on their *functional properties* that describe temporal system behaviors, safety or liveness conditions and so on.

This thesis looks into the extension of formal methods with *quantitative properties* of systems, leading towards scalable and efficient verification and synthesis *with quantitative properties*.

1.1 Quantitative analysis

The notion of correctness with functional properties is Boolean. The richness of modern systems justifies properties that are *quantitative*. These can be thought to extend functional properties, since in this case executions are assigned to a real-valued cost using a *cost model* that captures the quantitative dimension of interest.

Quantitative properties can reason about aspects such as quality measures, cost- and resource- consumption, distance metrics and the like [12, 14, 19, 66], which functional properties cannot easily express. For example, whether an arbiter grants every request is a functional property; But the promptness with which an arbiter grants requests is a quantitative property.

The analysis of quantitative properties of computing systems, or *quantitative analysis* in short, is an emerging area in automated formal analysis. It enables the formal reasoning about features such as timeliness, quality, reliability and so on. Quantitative properties have been used in several forms. These include probabilistic guarantees about the correctness of hardware or software [19, 66, 68]; use of distance-metrics to produce low-cost program repair [53, 60], generate feedback with few corrections in computer-aided education [84], and automated generation of questions of similar difficulty for MOOCs [9]; in planning for robots with resource-constraints [58], with hard and soft constraints where the soft constraints are expressed quantitatively [69, 90], and generate plans of higher quality [12, 29].

The cornerstones of formal analysis of quantitative properties of systems are (a). Verification of quantitative properties, and (b) Synthesis from quantiative properties. This thesis focuses on the two problems that form the fundamentals of these two cornerstones. These problems are *Quantiative inclusion* and *Solving quantitative games*, respectively, as detailed below:

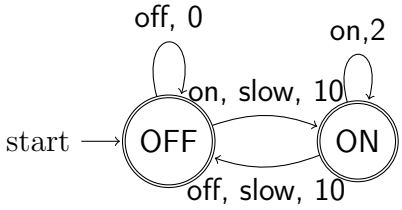
From "Verification of quantitative properties" to "Quantiative inclusion". In the verification of functional properties, the task is to verify if a system S satisfies a functional property P, possibly represented by a temporal logic [77]. Traditionally, S and P are interpreted as sets of (infinite-length) executions, and S is determined to

satisfy P if $S \subseteq P$. The task, however, can also be framed in terms of a comparison between executions in S and P. Suppose an execution w is assigned a cost of 1 if it belongs to the language of the system or property, and 0 otherwise. Then determining if $S \subseteq P$ amounts to checking whether the cost of every execution in S is less than or equal to its cost in P [21, 89].

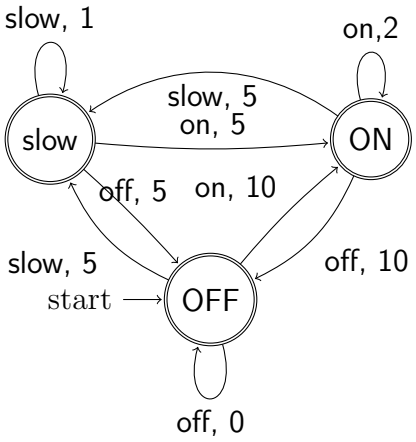
The verification of quantitative properties is an extension of the same from functional properties. In the quantitative scenario, executions in the system S and property P are assigned a real-valued cost, as opposed to a 0 or 1 cost. Then, as earlier, verification of quantitative properties amounts to checking whether the cost of every execution in S is less than or equal to its cost in P. When the system and property are modeled/represented by a finite-state (quantitative) abstraction, this problem is referred to as *quantitative inclusion* [39, 55]. Formally speaking, *given two finite-state (quantitative) abstractions S and P, quantitative inclusion determines whether the cost of every execution in S is less than or equal to its cost in P.*

For verification purposes, the systems are often modeled by finite-state (quantitative) abstractions. Even though properties are typically represented by quantitative logics, several of them can be converted to the aforementioned abstractions [46, 50]. Therefore, verification from quantitative properties reduces to quantitative inclusion. More generally, quantitative inclusion is applied to compare systems on their quantitative dimensions (Illustration in Fig 1.1), and in the analysis of rational behaviors in multi-agent systems with reward-maximizing agents [8, 23]. ■

From "Synthesis from quantitative properties" to "Solving quantitative games". Synthesis from a functional properties in an uncertain environment is perceived as a game between an uncontrollable environment and the system that



(a) Motor prototype 1



(b) Motor prototype 2

Figure 1.1 : Quantitative inclusion: Which motor prototype is more efficient? Weights on transitions indicate amount of energy consumed in that single step. Cost of an execution is the limit-average of costs along all transition on the execution, i.e., cost model is limit-average

we wish to designed. In this game, the environment and the system have the opposing objectives of guaranteeing that the resulting play flouts and satisfies the functional property, respectively. The system can be constructed if it is able to win the game against the environment. As earlier, synthesis from quantitative properties extends the later by assigning real-valued costs to the plays [29, 37, 58].

As a concrete instantiation, suppose we are to design a plan for a robot to patrol a grid without colliding into another (movable) robot present on the grid. Now, as our robot moves around the grid, it expends the charge on its battery, and will have to enter the recharge stations every now and then. Therefore, while planning for our robot, we need to guarantee that its battery charge never goes below level 0. Note that this criteria is a quantitative constraint.

Such problems of planning in an uncertain environment with quantitative constraints can be reduced to solving a min-max optimization on a two-player *quantitative graph game*. The two players refer to the system for which the plan is being designed, and an uncontrollable environment which is assumed to be adversarial to the objective of the system player. In these games, players take turns to pass a token along the *transition relation* between the states. As the token is pushed around, the play accumulates weights along the transitions using the underlying cost model $f : \mathbb{Z}^\omega \to \mathbb{R}$. One player maximizes the cost while the other minimizes it. ■

Both of the problems extend their functional counterparts. As a result, quantitative analysis inherits their high complexity and practical intractability [11, 20]. However, algorithms that solve over functional properties cannot be directly applied to solve with quantitative properties. Therefore, the developments in functional properties have limited impact on solving with qualitative properties. To this end, this thesis focuses on development of efficient and scalable solutions for quantitative analysis.

1.2 Challenges in quantitative analysis

This thesis begins with identifying two broad challenges that obstruct the theoretical understanding and algorithmic development in quantitative analysis.

Challenge 1. Lack of generalizability refers to the issue that solution approaches of problems in quantitative analysis are specialized to the underlying cost model. Different cost models deploy disparate techniques to solve. This disparity restricts the transfer of knowledge of solving problems over one cost model to another cost model. Furthermore, owning to the large variety of cost models, soon it will

become too cumbersome to digest progress made across cost models.

A concrete instantiation of this challenges appears in problems over *weighted* ω-*automata* [52]. Weighted ω-automaton [52] are a well-established finite-state (quantiative) abstraction used to model quantitative system, as illustrated in Fig 1.1. An execution is an infinite sequence of labels that arise from an infinite sequence of subsequent transitions. Weighted ω-automata assign a real-valued cost to all executions over the machine using a cost model $f : \mathbb{Z}^\omega \to \mathbb{R}$. The cost of an execution is assigned by applying the cost model f to the weight-sequence arising from transitions along the execution. In case the transition relation is non-deterministic, each execution may have multiple runs. In these cases, the cost of the execution is resolved by taking the infimum/supremum of cost along all runs.

In case of quantitative inclusion over weighted ω-automata, the lack of generalizable is apparent from the diversity in complexity-theoretic results. While quantitative inclusion is PSPACE-hard, there is ample variance in upper bounds. Quantitative inclusion is PSPACE-complete under limsup/liminf [39], undecidable for limit-average [50], and decidability is unknown for discounted-sum in general but its decidable fragments are EXPTIME [31, 39].

Yet another well-studied problem over weighted ω-automata is to determine if there exists an execution with cost that exceeds a constant threshold value [47, 52]. The problem finds applications in verification from quantitative logics [46, 50], planning with quantitative constraints in closed environments [69], low-cost program repair. The landscape is even less uniform on this problem. Known solutions involve diverse techniques ranging from linear-programming for discounted-sum [18] to negative-weight cycle detection for limit-average [61] to computation of maximum weight of cycles for limsup/liminf [38].

The question that this thesis asks is whether one can develop a unifying theory for quantitative analysis that generalizes across a variety of cost models. It would be too navie to expect that all cost models can be brought under a single theoretical framework. So, the natural question to ask is about boundary conditions over cost models that can and cannot be generalized. ■

Challenge 2. Separation-of-techniques refers to the inherent dichotomy in existing algorithms for problems in quantitative analysis. Most algorithms comprise of two phases. In the first phase, called the *structural phase*, the algorithm reasons about the structure of the quantitative system(s) using techniques from automata or graphs. In the second phase, called the *numerical phase*, the algorithm reasons about the quantitative dimension/cost model using numerical methods. The techniques used in both these phases are so unlike each other that it is difficult to solve them *in tandem*, and hence the phases have to be performed sequentially. The issue with this is it isn't uncommon to observe an exponential blow-up in the first phase. This poses a computational barrier to solvers of numerical methods in the second phase, that despite being of industrial strength are still unable to operate on exponentially larger inputs. Hence, this separation-of-techniques affects the scalability of algorithms in quantitative analysis.

As a concrete instantiation, consider the problem of planning in an uncertain environment under quantitative constraints. Here, the structural phase consists of the reduction from planning to the *quantitative graph game*, while the numerical phase consists of solving the min-max optimization on the game. In case of our robot planning example, the quantitative game has as many states as there are configurations of the grid. So, if the grid has a size of $n \times n$, then the quantitative graph game will

have $\mathcal{O}(n^4)$ states. Therefore, the size of the min-max optimization problems grows rapidly, and will limit scalability of planning.

The question this thesis asks is whether one can design an algorithmic approach for quantitative analysis that integrates *the two phases as opposed to the existing separation-of-techniques , and whether an integrated approach will lead to efficient and scalable solutions.*

Existing work on *probabilistic verification* presents initial evidence of the existence of integrated approaches [67, 76]. Probabilistic verification incorporates state-of-the-art symbolic data structures and algorithms based on BDDs (Binary Decision Diagrams) [34] or MTBDD (Multi-Terminal Binary Decision Diagrams). These Decision Diagrams can encode and manipulate both the the structural and numerical aspects of underlying problem at once, and hence are an integrated method. The disadvantage of Decision Diagrams is that their performance is unpredictable, as they are known to exhibit large variance in runtime and memory consumption. Therefore, existing integrated approaches fall short of wholesome improvement in the theory and practice of problems in quantitative analysis. ■

1.3 Thesis contributions

Existing solution approaches suffer from the lack of generalizability and separation-of-techniques, adversely impacting a clear theoretical understanding of quantitative analysis, and large-scale applicability due to limitations on scalability. This thesis contributes towards a novel theoretical framework that addresses both of the challenges, and demonstrates utility on problems of quantitative inclusion and solving quantitative games. The introduced framework, called *comparator automata* or *comparators* in short, builds on automata-theoretic foundations to generalize across a

variety of cost models. The crux of comparators is that they substitute the numerical analysis phase with automata-based methods, and hence naturally offer an integrated method for quantitative analysis. In all, we show that comparator-based algorithms have the advantages of generalizable results, yield complexity-theoretic and algorithmic development, and render practically scalable solutions.

The detailed contributions are as follows:

1. **Comparator automata**: The approach takes the view that the comparison of costs of system executions with the costs on other executions is the fundamental operation in quantitative reasoning, and hence that should be brought to the forefront. To this end, *comparator automata* as an automata-theoretic formulation of this form of comparison between weighted sequences. Specifically, comparator automata (comparators, in short), is a class of automata that read pairs of infinite weight sequences synchronously, and compare their costs in an online manner. We show that cost models such as limsup/liminf and discounted-sum with integer discount factors permit comparators that require af finite amount of memory, while comparators for limit-average require an infinite-amount of memory. We show that results and algorithms over weighted ω-automata and quantitative games generalize over cost models that permit comparators with finite memory. Lastly, since comparators are automata-theoretic, these algorithms are integrated in approach.

We illustrate further benefits of comparator-based approaches by investigating the discounted-sum cost model, which is a fundamental cost model in decision-making and planning domains including reinforcement learning, game theory, economics

2. **Quantitative inclusion over discounted-sum**: The decidability of quantita-

tive inclusion over discounted-sum, DS inclusion, is unknown when the discount factor is an integer. This has been an open problem for almost 15 years now. When the discount factor is an integer, DS inclusion is known to have an EXPTIME upper bound and PSPACE lower bound. Hence, its exact complexity is unknown.

Comparator-based arguments make resolutions towards both of these questions. We are able to prove that when the discount factor is an integer, then DS inclusion is indeed PSPACE-complete. When the discount factor is not an integer, then we are not able to resolve the decidability debate. However we are able to provide an anytime algorithm for the same, hence rendering a pragmatic solution.

Last but not the least, we demonstrate that by leveraging its automata-theoretic foundations, comparator-based algorithms for DS inclusion are more scalable and efficient in practice that existing methods.

3. **Quantitative games over discounted-sum**: The benefits of ω-regular and safety/co-safety properties of comparators for discounted-sum continue in quantitative games as well. We show that comparator-based solutions to quantitative games are more efficient in theory and more efficient in practice. In addition, they broaden the scope of quantitative games by extending with temporal goals.

1.4 Outline

Chapter 2 introduces the necessary notation and background, and should be treated as an index. The technical sections of this thesis have been split into three parts.

Part I lays down the theoretical foundations of *comparator automata*, our automata-

based technique to obtain integrated methods to solve problems in quantitative reasoning. The formal introduction of comparator automata, and utility of ω-regular comparators in designing generalizable solutions is in Chapter 3. Chapter 4 furthers the study by investigating aggregate functions that permit ω-regular functions.

Part II undertakes an investigation of discounted-sum automata, following the result that DS comparators are ω-regular iff their discount factor is an integer. The Part is split into three chapters. Chapter 5 and Chapter 6 are concerned with DS inclusion under integer discount factors. Safety and co-safety comparators for DS are introduced in Chapter 6. Chapter 7 is concerned with designing anytime algorithm for DS inclusion with non-integer discount factors.

Part III consists of Chapter 8. It studies the advantages of safety/co-safety automata comparator automata in discounted-sum games.

Last but not the least, the thesis concludes with a discussion of future directions in Chapter 9.

Chapter 2

Background

2.1 Automata and formal languages

Büchi automaton

Büchi automaton correspond to acceptors of words that require a finite-amount of memory. Formally, a (finite-state) *Büchi automaton* [87] is a tuple $\mathcal{A} = (S, \Sigma, \delta, \mathit{Init}, \mathcal{F})$, where S is a finite set of *states*, Σ is a finite *input alphabet*, $\delta \subseteq (S \times \Sigma \times S)$ is the *transition relation*, $\mathit{Init} \subseteq S$ is the set of *initial states*, and $\mathcal{F} \subseteq S$ is the set of *accepting states* [87].

A Büchi automaton is *deterministic* if for all states s and inputs a, $|\{s' | (s, a, s') \in \delta \text{ for some } s'\}| \leq 1$ and $|\mathit{Init}| = 1$. Otherwise, it is *nondeterministic*. A Büchi automaton is *complete* if for all states s and inputs a, $|\{s' | (s, a, s') \in \delta \text{ for some } s'\}| \geq 1$. For a word $w = w_0 w_1 \cdots \in \Sigma^\omega$, a *run* ρ of w is a sequence of states $s_0 s_1 \ldots$ s.t. $s_0 \in \mathit{Init}$, and $\tau_i = (s_i, w_i, s_{i+1}) \in \delta$ for all i. Let $\mathit{inf}(\rho)$ denote the set of states that occur infinitely often in run ρ. A run ρ is an *accepting run* if $\mathit{inf}(\rho) \cap \mathcal{F} \neq \emptyset$. A word w is an accepting word if it has an accepting run. The language $\mathcal{L}(\mathcal{A})$ of Büchi automaton $\mathcal{A}$ is the set of all words accepted by it. Languages accepted by these automata are called *ω-regular languages*. Büchi automata are known to be closed under set-theoretic union, intersection, and complementation [87]. For Büchi automata A and B, the *language-equivalence* and *language-inclusion* are whether $\mathcal{L}(A) \equiv \mathcal{L}(B)$ and $\mathcal{L}(A) \subseteq \mathcal{L}(B)$, resp.

Büchi pushdown automaton

Büchi pushdown automaton correspond to acceptors of words that require an infinite-amount of memory. A *Büchi pushdown automaton* [45] is a tuple $\mathcal{A} = (S, \Sigma, \Gamma, \delta, \mathit{Init}, Z_0, \mathcal{F})$, where S, Σ, Γ, and $\mathcal{F}$ are finite sets of *states*, *input alphabet*, *pushdown alphabet* and *accepting states*, respectively. $\delta \subseteq (S \times \Gamma \times (\Sigma \cup \{\epsilon\}) \times S \times \Gamma)$ is the *transition relation*, $\mathit{Init} \subseteq S$ is a set of *initial states*, $Z_0 \in \Gamma$ is the *start symbol*.

A *run* ρ on a word $w = w_0 w_1 \cdots \in \Sigma^\omega$ of a Büchi PDA $\mathcal{A}$ is a sequence of configurations $(s_0, \gamma_0), (s_1, \gamma_1) \ldots$ satisfying (1) $s_0 \in \mathit{Init}$, $\gamma_0 = Z_0$, and (2) $(s_i, \gamma_i, w_i, s_{i+1}, \gamma_{i+1}) \in \delta$ for all i. Büchi PDA consists of a *stack*, elements of which are the tokens Γ, and initial element Z_0. Transitions *push* or *pop* token(s) to/from the top of the stack. Let $\mathit{inf}(\rho)$ be the set of states that occur infinitely often in state sequence $s_0 s_1 \ldots$ of run ρ. A run ρ is an *accepting run* in Büchi PDA if $\mathit{inf}(\rho) \cap \mathcal{F} \neq \emptyset$. A word w is an *accepting word* if it has an accepting run. Languages accepted by Büchi PDA are called *ω-context-free languages* (ω-CFL).

Weighted ω-automaton

A *weighted automaton* [39, 72] over infinite words is a tuple $\mathcal{A} = (\mathcal{M}, \gamma, f)$, where $\mathcal{M} = (S, \Sigma, \delta, \mathit{Init}, S)$ is a Büchi automaton with all states as accepting, $\gamma : \delta \to \mathbb{Q}$ is a *weight function*, and $f : \mathbb{Q} \to \mathbb{R}$ is the *aggregate function* [39, 72].

Words and *runs* in weighted automata are defined as they are in Büchi automata. The *weight-sequence* of run $\rho = s_0 s_1 \ldots$ of word $w = w_0 w_1 \ldots$ is given by $wt_\rho = n_0 n_1 n_2 \ldots$ where $n_i = \gamma(s_i, w_i, s_{i+1})$ for all i. The *weight of a run* ρ, denoted by $f(\rho)$, is given by $f(wt_\rho)$. Here the *weight of a word* $w \in \Sigma^\omega$ in weighted automata is defined as $wt_\mathcal{A}(w) = sup\{f(\rho) | \rho \text{ is a run of } w \text{ in } \mathcal{A}\}$. In general, weight of a word can also be defined as the infimum of the weight of all its runs. By convention, if a

word $w \notin \mathcal{L}(\mathcal{M})$ its weight $wt_{\mathcal{A}}(w) = -\infty$.

2.2 Games over graphs

Games over graphs are popular in formal methods to study interactions between two players. More specifically, these are extensively used in the context of planning and synthesis. A graph game consists two players. It is played over a directed graph in which the states are partitioned between the two players. A play consists of the players moving a token along edges in the graph: The play begins in the initial state, and the token is pushed to the next state by the player that own the current state. This play goes on till infinitum. The outcome of which player wins a play is determined by an *acceptance condition* over the play itself. Reachability, safety, and parity are few examples of such conditions, and are described in detail below. In other cases, the plays could be associated with a quantitative value. In these cases, the winning criteria will be a quantitative property, as described later in the section.

Reachability and safety games

Both, *reachability* and *safety games*, are defined over the structure $G = (V = V_0 \uplus V_1, v_{init}, E, \mathcal{F})$ [87], where V, V_0, V_1, v_{init}, and E are defined as above. We assume that every state has at least one outgoing edge, i.e, $vE \neq \emptyset$ for all $v \in V$. The non-empty set of states $\mathcal{F} \subseteq V$ is called the *accepting* and *rejecting* states in reachability and safety games, resp.

A play $\rho = v_0 v_1 v_2 \dots$ is defined as earlier. A play is *winning for player* P_0 in a reachability game if no state in the play is an accepting state, and *winning for player* P_1 otherwise. The opposite holds in safety games. A play is winning for player P_0 if it visits the rejecting states, and winning for P_1 otherwise.

Strategies for players are defined as earlier as well. A strategy Π_i is winning for player P_i if strategies of the opponent player P_{1-i}, all resulting plays are winning for P_i. To *solve* a reachability/safety game refers to determining whether there exists a winning strategy for player P_1 in the game. Both reachability and safety games are solved in linear time to their size, i.e., $\mathcal{O}(|V| + |E|)$.

Parity games

A *parity game* is defined over the structure $G = (V = V_0 \uplus V_1, v_{init}, E, c)$ [87], where V, V_0, V_1, v_{init}, and E are defined as above. We assume that every state has at least one outgoing edge, i.e, $vE \neq \emptyset$ for all $v \in V$. The *coloring function* $c : V \to \mathbb{N}$ assigns a natural-valued *color* to states in the game.

Plays and strategies are defined as earlier. A color sequence of a play $c_\rho = c_0 c_1 c_2 \ldots$ where $c_i = c(v_i)$ for all $i \geq 0$. is Wlog, a play is said to be *winning* if the maximum color appearing infinitely often in its color sequence is even. As earlier, a strategy Π_i is winning for player P_i if strategies of the opponent player P_{1-i}, all resulting plays are winning for P_i. To *solve* a parity game refers to determining whether there exists a winning strategy for player P_1.

Quantitative games with complete information

A *quantitative graph with complete information game*, referred to as quantitative game in short, is defined over a structure $G = (V = V_0 \uplus V_1, v_{init}, E, \gamma, f)$. It consists of a directed graph (V, E), and a partition (V_0, V_1) of its set of states V. State v_{init} is the *initial state* of the game. vE designates the set $\{w \in V | (v, w) \in E\}$ to indicate the successor states of state $v \in V$. For convenience, we assume that every state has at least one outgoing edge, i.e, $vE \neq \emptyset$ for all $v \in V$. Each transition of the

game is associated with a *cost* determined by the *cost function* $\gamma : E \to \mathbb{Z}$. Finally, $f : \mathbb{Z}^\omega \to \mathbb{R}$ is the *aggregate function*.

A *play* of a game involves two players, denoted by P_0 and P_1, that form an infinite path by moving a token along the transitions as follows: At the beginning, the token is at the initial state. If the current position v belongs to V_0, then P_0 chooses the successor state w from vE; otherwise, if $v \in V_1$, then P_1 chooses the next state from vE. Formally, a play $\rho = v_0 v_1 v_2 \dots$ is an infinite sequence of states such that the first state $v_0 = v_{\mathsf{init}}$, and each pair of successive states is a transition, i.e., $(v_k, v_{k+1}) \in E$ for all $k \geq 0$. The *cost sequence* of a play ρ is the sequence of costs $w_0 w_1 w_2 \dots$ such that $w_k = \gamma((v_k, v_{k+1}))$ for all $i \geq 0$. Given the aggregate function $f : \mathbb{Z}^\omega \to \mathbb{R}$, the *cost of play* ρ, denoted $wt(\rho)$, is the aggregate function applied to the cost sequence, i.e., $wt(\rho) = f\rho$. W.l.o.g, we assume that the players have opposing objectives, one player attempts to maximize the cost from plays while the other attempts to minimize the cost.

A *strategy* for player P_i is a is a partial function $\Pi_i : V^* V_i \to V$ such that $v = \Pi_i(v_0 v_1 \dots v_k)$ belongs to $v_k E$. Intuitively, strategy Π_i directs the player P_i which state to go to next based on the history of the play such that it is consistent with the transitions. A player P_i is said to follow a strategy Π on a play ρ if for all k-length prefixes $v_0 v_1 \dots v_{k-1}$ of ρ if $v_{k-1} \in V_i$ then $v_k = \Pi(v_0 v_1 \dots v_{k-1})$.

Quantitative game with incomplete information

An *incomplete-information quantitative game* is a tuple $\mathcal{G} = (S, s_{\mathcal{I}}, O, \Sigma, \delta, \gamma, f)$, where S, O, Σ are sets of *states*, *observations*, and *actions*, respectively, $s_{\mathcal{I}} \in S$ is the *initial state*, $\delta \subseteq S \times \Sigma \times S$ is the *transition relation*, $\gamma : S \to \mathbb{Z} \times \mathbb{Z}$ is the *weight function*, and $f : \mathbb{Z}^\omega \to \mathbb{R}$ is the *aggregate function*.

The transition relation δ is *complete*, i.e., for all states p and actions a, there exists a state q s.t. $(p, a, q) \in \delta$. A *play* ρ is a sequence $s_0 a_0 s_1 a_1 \ldots$, where $\tau_i = (s_i, a_i, s_{i+1}) \in \delta$. The *observation of state* s is denoted by $O(s) \in O$. The *observed play* o_ρ of ρ is the sequence $o_0 a_0 o_1 a a_1 \ldots$, where $o_i = O(s_i)$. Player P_0 has incomplete information about the game $\mathcal{G}$; it only perceives the observation play o_ρ. Player P_1 receives full information and witnesses play ρ. Plays begin in the initial state $s_0 = s_{\mathcal{I}}$. For $i \geq 0$, Player P_0 selects action a_i. Next, player P_1 selects the state s_{i+1}, such that $(s_i, a_i, s_{i+1}) \in \delta$. The *weight of state* s is the pair of payoffs $\gamma(s) = (\gamma(s)_0, \gamma(s)_1)$. The *weight sequence* wt_i of player P_i along ρ is given by $\gamma(s_0)_i \gamma(s_1)_i \ldots$, and its payoff from ρ is given by $f(wt_i)$ for aggregate function f, denoted by $f(\rho_i)$, for simplicity. A play on which a player receives a greater payoff is said to be a *winning play* for the player. A strategy for player P_0 is given by a function $\alpha : O^* \to \Sigma$ since it only sees observations. Player P_0 follows strategy α if for all i, $a_i = \alpha(o_0 \ldots o_i)$. A strategy α is said to be a *winning strategy* for player P_0 if all plays following α are winning plays for P_0.

2.3 Aggregate functions

Let $A = A[0]A[1]\ldots$ be an infinite weight sequence. Let $A[i]$ denote the i-th element of the sequence, for all $i \geq 0$. Let $A[0, i-1]$ denote the i-length prefix $A[0] \ldots A[i-1]$ of a sequence A, for $i \geq 0$.

Discounted-sum aggregate function

Discounted-sum (DS) is a commonly appearing mode of aggregation which captures the intuition that weights incurred in the near future are more significant than those incurred later on [48]. The *discount-factor* is a rational-valued paramater that governs

the rate at which weights loose their significance in the DS.

Let $d > 1$ be the rational-valued discount-factor. Then the discounted-sum of an infinite weight sequence A w.r.t. discount-factor d, denoted by $DS(A, d)$, is defined as

$$DS(A, d) = \sum_{i=0}^{|A|-1} \frac{A[i]}{d^i} \tag{2.1}$$

DS is a preferred mode of aggregation across several domains including reinforcement learning [85], planning under uncertainty [78], and game-theory [75]. One reason behind its popularity is that DS is known to exist for all bounded infinite-length weight sequences, since its value is guaranteed to converge.

Limit-average aggregate function

The limit-average of an infinite is intuitively defined as is the point of convergence of the average of prefixes of the sequence.

Let $\mathsf{Sum}(A[0, n-1]) = \sum_{i=0}^{n-1} A[i]$ denote the sum of the n-length prefix of sequence A. Then intuitively, the limit-average of a sequence A should be defined as $\lim_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(A[0, n-1])$. However, this is not well-defined since $\frac{1}{n} \cdot \mathsf{Sum}(A[0, n-1])$ may not converge as $n \to \infty$. Therefore, limit-average is defined in terms of auxilary functions *limit-average infimum* and *limit-average supremum*.

The *limit-average infimum* of an infinite weight sequence A, denoted by $\mathsf{LimAvgInf}(A)$, is defined as

$$\mathsf{LimAvgInf}(A) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(A[0, n-1]) \tag{2.2}$$

The *limit-average supremum* of an infinite weight sequence A, denoted by $\mathsf{LimAvgSup}(A)$, is defined as

$$\mathsf{LimAvgSup}(A) = \limsup_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(A[0, n-1]) \tag{2.3}$$

Then, the limit-average of sequence A, denoted by $\mathsf{LimAvg}(A)$ is defined either as the limit-average infinum or supremum. Note, limit-average is well defined *only if* the limit-average infimum and limit-average supremum coincide, in which case limit-average is the same as limit-average infinum. In the other case, it is simply defined as either the limit-average supremum or infimum.

ω-regular aggregate function

The class of ω-regular aggregate function are those functions on which arithmetical operations can be performed on on automata [42].

Let $\beta \geq 2$ be an integer base, Let $\mathsf{Digit}(\beta) = \{0, \ldots, \beta - 1\}$ be its *digit set*. Let $x \in \mathbb{R}$, then there exist unique words $\mathsf{Int}(x, \beta) = z_0 z_1 \cdots \in \mathsf{Digit}(\beta)^* \cdot 0^\omega$ and $\mathsf{Frac}(x, \beta) = f_0 f_1 \ldots \notin \mathsf{Digit}(\beta)^* \cdot (\beta - 1)^\omega$ such that $|x| = \sum_{i=0}^{\infty} \beta^i \cdot z_i + \sum_{i=0}^{\infty} \frac{f_i}{\beta^i}$. Thus, z_i and f_i are respectively the i-th least significant digit in the base β representation of the integer part of x, and the i-th most significant digit in the base β representation of the fractional part of x. Then, the real-number $x \in \mathbb{R}$ in base β is represented by $\mathsf{rep}(x, \beta) = \mathsf{sign} \cdot (\mathsf{Int}(x, \beta), \mathsf{Frac}(x, \beta))$, where $\mathsf{sign} = +$ if $x \geq 0$, $\mathsf{sign} = -$ if $x < 0$, and $(\mathsf{Int}(x, \beta), \mathsf{Frac}(x, \beta))$ is the interleaved word of $\mathsf{Int}(x, \beta)$ and $\mathsf{Frac}(x, \beta)$. Clearly, $x = \mathsf{sign} \cdot |x| = \mathsf{sign} \cdot (\sum_{i=0}^{\infty} \beta^i \cdot z_i + \sum_{i=0}^{\infty} \frac{f_i}{\beta^i})$. For all integer $\beta \geq 2$, we denote the alphabet of representation of real-numbers in base β by $\mathsf{AlphaRep}(\beta)$.

Definition 2.1 (Aggregate function automaton, ω-Regular aggregate function) Let Σ be a finite set, and $\beta \geq 2$ be an integer-valued base. A Büchi automaton $\mathcal{A}$ over alphabet $\Sigma \times \mathsf{AlphaRep}(\beta)$ is an *aggregate function automata of type* $\Sigma^\omega \to \mathbb{R}$ if the following conditions hold:

- For all $A \in \Sigma^\omega$, there exists at most one $x \in \mathbb{R}$ such that $(A, \mathsf{rep}(x, \beta)) \in \mathcal{L}(\mathcal{A})$,

and

- For all $A \in \Sigma^\omega$, there exists an $x \in \mathbb{R}$ such that $(A, \mathsf{rep}(x, \beta)) \in \mathcal{L}(\mathcal{A})$

Σ and $\mathsf{AlphaRep}(\beta)$ are the *input* and *output* alphabets, respectively. An aggregate function $f : \Sigma^\omega \to \mathbb{R}$ is *ω-regular* under integer base $\beta \geq 2$ if there exists an aggregate function automaton $\mathcal{A}$ over alphabet $\Sigma \times \mathsf{AlphaRep}(\beta)$ such that for all sequences $A \in \Sigma^\omega$ and $x \in \mathbb{R}$, $f(A) = x$ iff $(A, \mathsf{rep}(x, \beta)) \in L(\mathcal{A})$.

2.4 Quantitative inclusion

Definition 2.2 (Quantitative inclusion) Let P and Q be weighted ω-automata with the *same* aggregate function f. The *strict quantitative inclusion problem*, denoted by $P \subset_f Q$, asks whether for all words $w \in \Sigma^\omega$, $wt_P(w) < wt_Q(w)$. The *non-strict quantitative inclusion problem*, denoted by $P \subseteq_f Q$, asks whether for all words $w \in \Sigma^\omega$, $wt_P(w) \leq wt_Q(w)$.

Quantitative inclusion, strict and non-strict, is PSPACE-complete for limsup and liminf [39], and undecidable for limit-average [50]. For discounted-sum with integer discount-factor it is in EXPTIME [31, 39], and decidability is unknown for rational discount-factors

Applications of quantitative inclusion are verification from quantitative logics, verification of reward-maximizing agents in multi-agent systems such as online economic protocols, auctions systems and so on, and in comparing systems based on a quantitative dimension.

2.5 Solving quantitative games

Games with complete information

Several problems in planning and synthesis with quantitative properties are formulated into an *analysis problem* over a quantiative game [37]. *Optimization* over such games is a popular choice of analysis in literature.

Definition 2.3 (Optimization problem) Given a quantitative graph game G, the optimization problem is to compute the optimal cost from all possible plays from the game, under the assumption that the objectives of P_0 and P_1 are to maximize and minimize the cost of plays, respectively.

Zwick and Patterson have shown that the optimization problem is pseudo-polynomial, and that the optimal cost can be obtained via memoryless strategies for both players [93]. They also show that a VI algorithm will converge to the optimal cost. However, a thorough worst-case analysis of VI is missing.

We argue that the optimal solution may not be required in several tasks. For instance, a solution with *minimal* battery consumption may be replaced by one that operates *within* the battery life. Hence, many tasks can be reformulated into an alternative form of analysis in which the problem is to search for a solution the adheres to a given threshold constraint [1]. We call this the *satisficing problem*.

Definition 2.4 (Satisficing problem) Given a quantitative graph game G and a threshold value $v \in \mathbb{Q}$, the *satisficing problem* is to determine whether player P_1 has a strategy such that for all possible resulting plays of the game, the cost of all plays is less than (or $\leq$) to the threshold v, assuming that the objectives of P_0 and P_1 are to maximize and minimize the cost of plays, respectively.

In several tasks one is interested in combining quantitative games with a temporal goal. For quantitative games, it is known that an optimal solution may not exist when combined with temporal goals [41]

Games with incomplete information

The main problem in games with incomplete information is to determine whether a player has a winning strategy. If so, one is interested in generating it.

Part I

Theoretical framework

This part lays down the theoretical foundations of *comparator automata*, or *comparators* in short.

Chapter 3 formally introduces comparator automata, our automata-based technique to obtain integrated methods to solve problems in quantitative reasoning, as opposed to separation-of-techniques used so far. We show that generalizable solutions for quantitative inclusion and solving quantitative games can be designed for all aggregate functions for which the comparator is represented by a Büchi automata.

Chapter 4 studies and constructs comparator automata commonly occurring aggregate functions, namely discounted-sum and limit-average.

Chapter 3

Comparator automata: Foundations of a generalizable and integrated theory for quantitative reasoning

The landscape of algorithms, complexity, tools and techniques for the quantitative analysis of systems is wide and non-uniform. A part of the reason is that there are several aggregate functions, and for each one of those a different form of reasoning is applied. For instance, consider the problem of quantitative inclusion for aggregate function f, called *f-inclusion* in short. In case of quantitative inclusion, even the complexity-theoretic results are diverse. While quantiative inclusion is PSPACE-hard, there is ample variance in upper bounds. Quantitative inclusion is PSPACE-complete under limsup/liminf [39], undecidable for limit-average [50], and decidability is unknown for discounted-sum in general but its decidable fragments are EXPTIME [31,39]. In view of these vast differences, the aforementioned landscape could benefit from a unified theory that *generalizes* across aggregate functions. This chapter contributes towards just that: A unified theory for quantitative reasoning that generalizes across a wide class of aggregate functions.

To this end, we take the view that the notion of *comparisons* between systems runs or inputs is central to formal methods especially to quantitative analysis, and hence comparison should be brought to the forefront. To see why our view holds, first consider the classical *model checking problem* of verifying if a system S satisfies a linear-time temporal specification P [44]. Traditionally, this problem is phrased

language-theoretically: S and P are interpreted as sets of (infinite) words, and S is determined to satisfy P if $S \subseteq P$. The problem, however, can also be framed in terms of a *comparison* between words in S and P. Suppose a word w is assigned a weight of 1 if it belongs to the language of the system or property, and 0 otherwise. Then determining if $S \subseteq P$ amounts to checking whether the weight of every word in S is less than or equal to its weight in P [21]. The ubiquity of comparisons becomes more pronounced in quantitative analysis: Firstly, because every system execution is assigned a real-valued cost. W.l.o.g, we can assume that the cost model is an *aggregate function* $f : \mathbb{Z}^\omega \to \mathbb{R}$. The cost of an execution is related to the aggregate function f applied to the weight-sequence corresponding to the execution; Secondly, because problems in quantitative analysis reduce to comparing the cost of executions to a constant value (such as in quantitative games), or more generally to the cost of another execution (as in quantitative inclusion).

Keeping comparisons at the center, we introduce a language-theoretic/automata-theoretic formulation of the comparison between weighted sequences. Specifically, in Section 3.1 we introduce *comparator automata* (*comparators*, in short), a class of automata that read pairs of infinite weight sequences synchronously, and compare their aggregate values in an online manner. Formally, a *comparator automata for aggregate function f, relation R, and upper bound $\mu > 0$* is an automaton that accepts a pair $(A, B) \in (\Sigma \times \Sigma)^\omega$ of sequences of bounded integers, where $\Sigma = \{-\mu, \mu - 1, \ldots, \mu\}$, iff $f(A)\ \mathsf{R}\ f(B)$, where $\mathsf{R} \in \{>, <, \geq, \leq, \neq =\}$ is an inequality or equality relation. A comparator could be finite-state or (pushdown) infinite-state. We say a comparator is *ω-regular* if it is finite-state and accepts by the Büchi condition. Similarly, we say a comparator is *ω-pushdown* if it is an ω-context free automaton.

The central result of this chapter is that one can design generalizable solutions

for all aggregate functions that permit an ω-regular comparator (Section 3.2). We illustrate this point by designing algorithms for quantitative inclusion (Section 3.2.1), and solving quantitative games under perfect and imperfect information (Section 3.2.2 and Section 3.2.3, respectively) for an aggregate function $f : \mathbb{Z} \to \mathbb{R}$ such that the comparator for f is ω-regular. What enables these generalizable results is the fact that since Büchi automata are closed under all set-theoretic operations, ω-regular comparators can easily be operated on in the generic algorithms that we design. Since ω-pushdown automata are not closed under all set-theoretic operations, one cannot design such generic algorithms with ω-pushdown comparators. In the next chapter (Chapter 4), we give concrete instances of aggregate functions which permit ω-regular or ω-pushdown comparators.

Another benefit of comparators has to do with the separation-of-techniques challenge of quantitative analysis. Recall from Chapter 1 that separation-of-techniques refers to the inherent dichotomy in existing algorithms for problems in quantitative analysis. Most algorithms comprise of two phases. In the first phase, called the *structural analysis phase*, the algorithm reasons about the structure of the quantitative system(s) using techniques from automata or graphs. In the second phase, called the *numerical analysis phase*, the algorithm reasons about the quantitative dimension/cost model using numerical methods. The techniques used in both these phases are so unlike each other that they cannot be solved *in tandem*. Hence the phases have to be performed sequentially, hence affecting their scalability.

The advantage our automata-based formulation of comparators in this aspect is that comparators reduce the numerical problem of comparison of aggregation of weight sequences into one of membership in an automaton. This way, enabling both phases, the structural phase and numerical phase, of quantitative analysis to be per-

formed using automata-based techniques. Subsequently, creating an opportunity to design *integrated methods* as opposed to separation-of-techniques methods for problems in quantitative analysis. Infact, the generalizable algorithms that we design when the aggregate function permits ω-regular comparators are based on reducing the problems in quantitative analysis to those in qualitative analysis. In particular, quantitative inclusion is reduced to *language inclusion*, and quantitative games under perfect and imperfect information are both reduced to solving *parity games*.

As a result, our novel comparator automata framework not only results in generalizable algorithms for problems in quantitative analysis, but the algorithms designed using comparators are inherently integrated by approach. Thereby, comparators resolve both challenges in quantitative analysis.

3.1 Comparison language and comparator automata

This section introduces comparison languages and comparator automata as a a class of langauges/automata that can read pairs of weight sequences synchronously and establish an equality or inequality relationship between these sequences. Formally, we define:

Definition 3.1 (Comparison language) Let Σ be a finite set of rational numbers, and $f : \mathbb{Q}^\omega \to \mathbb{R}$ denote an aggregate function. A *comparison language for aggregate function f with inequality or equality relation* $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$ is the language over the alphabet $\Sigma \times \Sigma$ that accepts a pair of infinite length sequences (A, B) iff $f(A)\ \mathsf{R}\ f(B)$ holds.

Definition 3.2 (Comparator automata) Let Σ be a finite set of rational numbers, and $f : \mathbb{Q}^\omega \to \mathbb{R}$ denote an aggregate function. A *comparator automaton for*

aggregate function f with inequality or equality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$ is the automaton that accepts the comparison language for f over alphabet $\Sigma \times \Sigma$ with relation R.

From now on, unless mentioned otherwise, we assume that all weight sequences are bounded integer sequences. First, the boundedness assumption is justified since the set of weights forming the alphabet of a comparator is bounded. In particular, Σ is bounded as it is a finite set. Second, for all aggregate functions considered in this thesis, the result of comparison of weight sequences is preserved by a uniform linear transformation that converts rational-valued weights into integers; justifying the integer assumption. Hence, now onward, we will define a comparison language w.r.t an *upper bound* $\mu > 0$ so that $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu\}$.

3.1.1 ω-regular comparator

When the comparison language for an aggregate function and a relation can be represented by a Büchi automaton, we refer to the comparison language and the comparator automata as *ω-regular*. Note that we **do not** refer to the corresponding aggregate function as ω-regular. This is because existing literature already defines ω-regular aggregate functions [42]. Section 4.4 will explore the relationship between ω-regular comparison languages/comparators and aggregation functions. ω-regular comparison languages benefit from closure-properties of Büchi automata. As a result, ω-regular comparison languages exhibit the following closure properties:

Theorem 3.1 (Closure) *Let $\mu >$ be an upper bound such that $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu\}$, and $f : \mathbb{Z}^\omega \to \mathbb{R}$ be an aggregate function. Let $\mathsf{R} \in \{\leq, \geq, <, >\}$ be an inequality relation.*

1. *If the comparison language for f over $\Sigma \times \Sigma$ with inequality relation R is ω-regular, then the comparison language for f over $\Sigma \times \Sigma$ with all inequality and equality relations $\mathsf{R}' \in \{\leq, \geq, <, >, =, \neq\}$ is ω-regular*

   aggregate function f is ω-regular.

2. *The comparison language for f over $\Sigma \times \Sigma$ with relation $=$ is ω-regular iff the comparison language for f over $\Sigma \times \Sigma$ with relation $\neq$ is ω-regular.*

3. *If the comparison language for f over $\Sigma \times \Sigma$ with relation R is not ω-regular, then the comparison language for f over $\Sigma \times \Sigma$ with all inequality relations $\mathsf{R}' \in \{\leq, \geq, <, >\}$ is not ω-regular.*

Proof 1 Item 1. W.l.o.g, let us assume that the comparison language for f with $\leq$ is ω-regular. This means that automaton representing the mentioned language is a Büchi automaton, say it is $\mathcal{A}$. Now, since Büchi automaton are closed under complementation, the automaton for the complementation of $\mathcal{A}$ is also a Büchi automaton. But note that the language of the complementation of $\mathcal{A}$ is the comparison language for $>$. Therefore, comparison language for $>$ is also ω-regular. Now, it is easy to show that (A, B) is a word in the comparison language for $\leq$ iff (B, A) is a word in the comparison language of $\geq$. Now the Büchi automaton for comparison language for $\geq$ can be constructed from $\mathcal{A}$ by swapping the alphabet. Therefore, the comparison language for $\geq$ is also ω-regular. By the same reason as above, if the comparison language for $\geq$ is ω-regular, then the comparison language for $<$ is also ω-regular. Finally, since the intersection of Büchi automaton is also a Büchi automaton, the Büchi automaton for comparison language of $=$ can be obtained by intersecting those for $\leq$ and $\geq$. Lastly, $\leq$ can be obtained by complementing that for $\neq$.

Hence, one can prove that if the comparison language for any one inequality

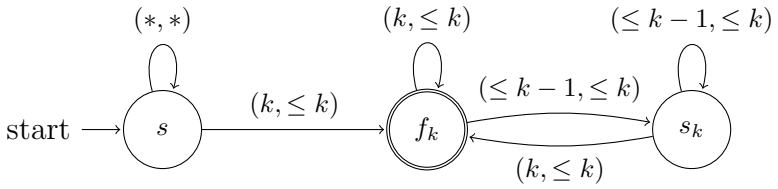


Figure 3.1 : Snippet of the Limsup comparator. State f_k is an accepting state. Automaton $\mathcal{A}_k$ accepts (A, B) iff $\mathsf{LimSup(A)} = k$, $\mathsf{LimSup(B)} \leq k$. $*$ denotes $\{-\mu, -\mu+1, \ldots, \mu\}$, $\leq m$ denotes $\{-\mu, -\mu+1, \ldots, m\}$

relation is ω-regular is sufficient to show that the comparison language for all relations is ω-regular. Item 2 and Item 3 can similarly be shown using closure properties of Büchi automaton. ■

It is worth mentioning that Item(1) is a means to demonstrate whether an aggregate function is ω-regular. Chapter 4 will give concrete examples of aggregate function that are ω-regular. We issue a concrete example below to illustrate these concepts.

Illustrative example: Limsup aggregate function

We explain comparison languages and comparator automata through an example. The aggregate function we consider is the *limit supremum* function. Loosely speaking, the limit supremum function determines the maximally appearing value appearing in an infinite sequence. We will construct a Büchi automaton corresponding to its comparator automaton for an inequality relation to show that its comparison language/comparator is ω-regular (Lemma 3.1).

The *limit supremum* (limsup, in short) of a bounded, integer sequence A, denoted by $\mathsf{LimSup(A)}$, is the largest integer that appears infinitely often in A. For a given

upper bound $\mu > 0$, The *limsup comparison language for relation* $\geq$ is a language over $\Sigma \times \Sigma$, where $\Sigma = \{-\mu, -\mu+1, \ldots, \mu\}$, that accepts the pair (A, B) of sequences iff $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$. We will show that the limsup comparison languages are ω-regular for all relations. To this end, we prove that the *limsup comparator for* $\geq$ is a Büchi automaton.

The working of the limsup comparator for relation $\geq$ is based on *guessing* the limsup of sequences A and B, and then *verifying* that $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$. This can be encoded using a non-deterministic Büchi automaton, as partially illustrated by $\mathcal{A}_k$ in Fig. 3.1. More specifically, $\mathcal{A}_k$ is the basic building block of the limsup comparator for relation $\geq$. Automaton $\mathcal{A}_k$ accepts pair (A, B) of number sequences iff $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(B)} \leq k$, for integer k. (Lemma 3.1).

Lemma 3.1 *Let A and B be integer sequences bounded by μ. Büchi automaton $\mathcal{A}_k$ (Fig. 3.1) accepts (A, B) iff $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$.*

Proof 2 Let (A, B) have an accepting run in $\mathcal{A}_k$. We show that $\mathsf{LimSup(A)} = k \geq \mathsf{LimSup(B)}$. The accepting run visits state f_k infinitely often. Note that all incoming transitions to accepting state f_k occur on alphabet $(k, \leq k)$ while all transitions between states f_k and s_k occur on alphabet $(\leq k, \leq k)$, where $\leq k$ denotes the set $\{-\mu, -\mu+1, \ldots, k\}$. So, the integer k must appear infinitely often in A and all elements occurring infinitely often in A and B are less than or equal to k. Therefore, if (A, B) is accepted by $\mathcal{A}_k$ then $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(B)} \leq k$, and $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$.

Conversely, let $\mathsf{LimSup(A)} = k > \mathsf{LimSup(B)}$. We prove that (A, B) is accepted by $\mathcal{A}_k$. For an integer sequence A when $\mathsf{LimSup(A)} = k$ integers greater than k can occur only a finite number of times in A. Let l_A denote the index of the last

occurrence of an integer greater than k in A. Similarly, since $\mathsf{LimSup(B)} \leq k$, let l_B be index of the last occurrence of an integer greater than k. Therefore, for sequences A and B integers greater than k will not occur beyond index $l = max(l_A, l_B)$. Büchi automaton $\mathcal{A}_k$ (Fig. 3.1) non-deterministically determines l. On reading the l-th element of input word (A, B), the run of (A, B) exits the start state s and shifts to accepting state f_k. Note that all runs beginning at state f_k occur on alphabet (a, b) where $a, b \leq k$. Therefore, (A, B) can continue its infinite run even after transitioning to f_k. To ensure that this is an accepting run, the run must visit accepting state f_k infinitely often. But this must be the case, since k occurs infinitely often in A, and all transitions on (k, b), for all $b \leq k$, transition into state f_k. Hence, for all integer sequences A,B bounded by μ, if $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$, the automaton accepts (A, B). ■

Theorem 3.2 *The comparison language for the limsup aggregate function is ω-regular for all relations $\mathsf{R} \in \{\leq, \geq, <, >, =, \neq\}$.*

Proof 3 Let $\mu > 0$ be the upper bound, then $\Sigma = \{-\mu, -\mu+1, \ldots, \mu\}$. To prove the above, we show that the limsup comparator with $\geq$ over alphabet $\Sigma \times \Sigma$ is ω-regular. This is sufficient due to Theorem 3.1-Item 1.

Observe that the union of Büchi automata $\mathcal{A}_k$ (Fig 3.1, Theorem 3.1) for $k \in \{-\mu, -\mu + 1, \ldots, \mu\}$ corresponds to the coveted limsup comparator for relation $\geq$. Since the union of Büchi automata is also a Büchi automata, this implies that limsup comparator for $\geq$ is ω-regular. ■

The *limit infimum* (liminf, in short) of an integer sequence is the smallest integer that appears infinitely often in it; its comparators will have a similar construction to their limsup counterparts. Hence, the comparison language for the liminf aggregate

function is also ω-regular for all relations.

3.1.2 ω-pushdown comparator

When the comparison language for an aggregate function and a relation can be represented by a Büchi pushdown automaton (and not a Büchi automaton), then the comparison language and comparator automata are referred to as *ω-pushdown*.

Unlike ω-regular comparison languages, ω-pushdown comparison languages may not exhibit closure properties since the underlying Büchi pushdown automaton are not closed under several operations such as complementation and intersection.

The next chapter (Chapter 4) will illustrate an example of an aggregate function for which the comparison language is ω-pushdown.

3.2 Generalizability with ω-regular comparators

This section illustrates the generalizability with ω-regular comparators. We demonstrate this over the problems of quantitative inclusion, and solving quantitative games under perfect and imperfect information. By virtue of the automata-based nature of comparator automata, all algorithms we design are integrated in approach.

3.2.1 Quantitative inclusion

The analysis of quantitative dimensions of computing systems such as cost, resource consumption, and distance metrics [14, 19, 66] has been studied thoroughly to design efficient computing systems. Cost-aware program-synthesis [29, 37] and low-cost program-repair [60] have found compelling applications in robotics [58, 69], education [53], and the like. *Quantitative verification* facilitates efficient system design

by automatically determining if a system implementation is more efficient than a specification model.

At the core of quantitative verification lies the problem of *quantitative inclusion* which formalizes the goal of determining which of two given systems is more efficient [39, 55, 72]. In quantitative inclusion, quantitative systems are abstracted as weighted ω-automata [16, 52, 73]. Recall, a run in a weighted ω-automaton is associated with a sequence of weights. The quantitative dimension of these runs is determined by the weight of runs, which is computed by taking an aggregate of the run's weight sequence. The problem of quantitative inclusion between two weighted ω-automata determines whether the weight of all words in one automaton is less that (or equal to) that in the other automaton. It can be thought of as the quantitative generalization of (qualitative) language inclusion.

In the chapter's preface, we saw how the solution approaches and complexity for f-inclusion vary with differences in the aggregate function f. This section presents a generic algorithm (Algorithm 1) to solve quantitative inclusion for function which permits ω-regular comparators. This section focuses on the *non-strict* variant of quantitative inclusion. Strict quantitative inclusion is similar, hence has been skipped.

Given weighted ω-automata P and Q with an aggregate function f, let whether P is non-strictly f-included in Q be denoted by $P \subseteq_f Q$. For sake of succinctness, we simply say *inclusion with an ω-regular comparator* to mean *f-inclusion where the function f permits an ω-regular comparator.*

Running example

The following running example is used to walk us through the steps of InclusionRegular, the generic algorithm for inclusion with ω-regular comparators presented in Algo-

(a) Weighted automaton P

(b) Weighted automaton Q

Figure 3.2 : Algorithm InclusionRegular: Running example

rithm 1.

Let weighted ω-automata P and Q be as illustrated in Fig. 3.2a-3.2b with the limsup aggregate function. From Theorem 3.2 we know that the limsup comparator $\mathcal{A}^{\leq}_{\mathsf{LS}}$ for $\leq$ is ω-regular. Therefore, the generic algorithm we describe will apply to the example.

The word $w = a^\omega$ has one run $\rho_1^P = p_1 p_2^\omega$ with weight sequence $wt_1^P = 1^\omega$ in P and two runs $\rho_1^Q = q_1 q_2^\omega$ with weight sequence $wt_1^Q = 0, 1^\omega$ and run $\rho_2^Q = q_1 q_2^\omega$ with weight sequence $wt_2^Q = 2, 1^\omega$. Clearly, $wt_P(w) \leq wt_Q(w)$. Therefore $P \subseteq_f Q$.

This section describes Algorithm 1 for inclusion for ω-regular comparators with the motivating example as a runnign example. Intuitively, the algorithm must be able to identify that for run ρ_1^P of w in P, there exists a run ρ_2^Q in Q s.t. (wt_1^P, wt_2^Q) is accepted by the limsup comparator for $\leq$.

Algorithm description and analysis

Key ideas A run ρ_P in P on word $w \in \Sigma^\omega$ is said to be *dominated* w.r.t $P \subseteq_f Q$ if there exists a run ρ_Q in Q on the same word w such that $wt_P(\rho_P) \leq wt_Q(\rho_Q)$. $P \subseteq_f Q$ holds if for every run ρ_P in P is dominated w.r.t. $P \subseteq_f Q$.

The central construction of InclusionRegular is a Büchi automaton *Dom* that consists of exactly the domianted runs of P w.r.t $P \subseteq_f Q$. Then, InclusionRegular returns

Algorithm 1 InclusionRegular$(P, Q, \mathcal{A}_f)$, Is $P \subseteq_f Q$?

1: **Input:** Weighted ω-automata P and Q over function f, and ω-regular comparator $\mathcal{A}_f$ for function f for the inequality $\leq$

2: **Output:** True if $P \subseteq_f Q$, False otherwise

3: $\hat{P} \leftarrow$ AugmentWtAndLabel(P)

4: $\hat{Q} \leftarrow$ AugmentWtAndLabel(Q)

5: $\hat{P} \times \hat{Q} \leftarrow$ MakeProduct$(\hat{P}, \hat{Q})$

6: $DomProof \leftarrow$ Intersect$(\hat{P} \times \hat{Q}, \mathcal{A}_{\succeq})$

7: $Dom \leftarrow$ FirstProject$(DomProof)$

8: **return** $\hat{P} \equiv Dom$

True iff Dom contains all runs of P. In order to construct Dom, the algorithm first constructs another Büchi automaton $DomProof$ that accepts word (ρ_P, ρ_Q) iff ρ_P and ρ_Q are runs of the same word in P and Q respectively, and $wt_P(\rho_P) \leq wt_Q(\rho_Q)$ i.e. if w_P and w_Q are weight sequence of ρ_P and ρ_Q, respectively, then (w_P, w_Q) is present in the ω-regular comparator $\mathcal{A}_f^{\leq}$ for aggregate function f with relation $\leq$. The projection of $DomProof$ on runs of P results in Dom.

Algorithm details For sake a simplicity, we assume that every word present in P is also present in Q i.e. $P \subseteq Q$ (qualitative inclusion). InclusionRegular has three steps: (a). UniqueId (Lines 1-4): Enables unique identification of runs in P and Q through *labels*. (b). Compare (Lines 5-4): Compares weight of runs in P with weight of runs in Q, and constructs Dom. (c). Ensure (Line 5): Ensures if all runs of P are diminished.

1. UniqueId: AugmentWtAndLabel transforms weighted ω-automaton $\mathcal{A}$ into Büchi

automaton $\hat{\mathcal{A}}$ by converting transition $\tau = (s, a, t)$ with weight $\gamma(\tau)$ in $\mathcal{A}$ to transition $\hat{\tau} = (s, (a, \gamma(\tau), l), t)$ in $\hat{\mathcal{A}}$, where l is a unique label assigned to transition τ. The word $\hat{\rho} = (a_0, n_0, l_0)(a_1, n_1, l_1)\cdots \in \hat{A}$ iff run $\rho \in \mathcal{A}$ on word $a_0 a_1 \ldots$ with weight sequence $n_0 n_1 \ldots$. Labels ensure bijection between runs in $\mathcal{A}$ and words in $\hat{\mathcal{A}}$. Words of $\hat{A}$ have a single run in $\hat{A}$. Hence, transformation of weighted ω-automata P and Q to Büchi automata $\hat{P}$ and $\hat{Q}$ enables disambiguation between runs of P and Q (Line 1-4).

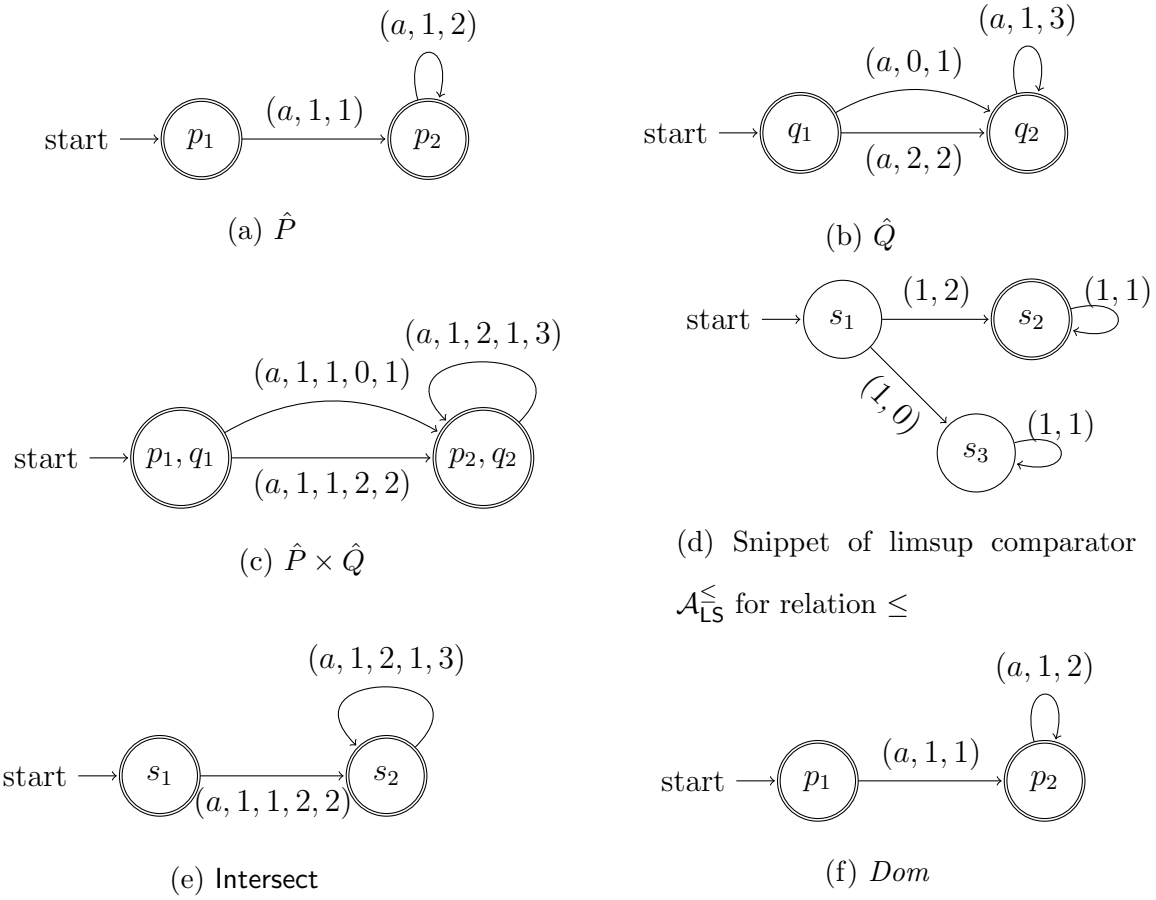
The corresponding $\hat{A}$ for weighted ω-automata P and Q from Figure 3.2a- 3.2b are given in Figure 3.3a- 3.3b respectively.

2. Compare: The output of this step is the Büchi automaton *Dom*, that contains the word $\hat{\rho} \in \hat{P}$ iff ρ is a dominated run in P w.r.t $P \subseteq_f Q$ (Lines 5-4).

   MakeProduct$(\hat{P}, \hat{Q})$ constructs $\hat{P} \times \hat{Q}$ s.t. word $(\hat{\rho_P}, \hat{\rho_Q}) \in \hat{P} \times \hat{Q}$ iff ρ_P and ρ_Q are runs of the same word in P and Q respectively (Line 5). Concretely, for transition $\hat{\tau_{\mathcal{A}}} = (s_{\mathcal{A}}, (a, n_{\mathcal{A}}, l_{\mathcal{A}}), t_{\mathcal{A}})$ in automaton $\mathcal{A}$, where $\mathcal{A} \in \{\hat{P}, \hat{Q}\}$, transition $\hat{\tau_P} \times \hat{\tau_Q} = ((s_P, s_Q), (a, n_P, l_P, n_Q, l_Q), (t_P, t_Q))$ is in $\hat{P} \times \hat{Q}$, as shown in Figure 3.3c.

   Intersect intersects the weight components of $\hat{P} \times \hat{Q}$ with comparator $\mathcal{A}_f^{\leq}$ (Line 3). The resulting automaton *DomProof* accepts word $(\hat{\rho_P}, \hat{\rho_Q})$ iff $f(\rho_P) \leq f(\rho_Q)$, and ρ_P and ρ_Q are runs on the same word in P and Q respectively. The result of Intersect between $\hat{P} \times \hat{Q}$ with the limsup comparator $\mathcal{A}_{\mathsf{LS}}^{\leq}$ for relation $\leq$ (Figure 3.3d) is given in Figure 3.3e.

   The projection of *DomProof* on the words of $\hat{P}$ returns *Dom* which contains the word $\hat{\rho_P}$ iff ρ_P is a dominated run in P w.r.t $P \subseteq_f Q$ (Line 4), as shown in Figure 3.3f.



(a) $\hat{P}$ (b) $\hat{Q}$ (c) $\hat{P} \times \hat{Q}$ (d) Snippet of limsup comparator $\mathcal{A}^{\leq}_{\mathsf{LS}}$ for relation $\leq$ (e) Intersect (f) *Dom*

Figure 3.3 : Algorithm InclusionRegular: Steps on the running example

3. Ensure: $P \subseteq_f Q$ iff $\hat{P} \equiv Dom$ (qualitative equivalence) since $\hat{P}$ consists of all runs of P and Dom consists of all domianted runs w.r.t $P \subseteq_f Q$ (Line 5).

Algorithm details We now prove the correctness of the algorithm and determine its complexity. We begin with a proof of correctness.

Lemma 3.2 *Büchi automaton Dom consists of all domianted runs in P w.r.t $P \subseteq_f Q$.*

Proof 4 Let $\mathcal{A}_f^{\leq}$ be the comparator for ω-regular aggregate function f and relation $\leq$ s.t. $\mathcal{A}_f$ accepts (A, B) iff $f(A) \leq f(B)$. A run ρ over word w with weight sequence wt in P (or Q) is represented by the unique word $\hat{\rho} = (w, wt, l)$ in $\hat{P}$ (or $\hat{Q}$) where l is the unique label sequence associated with each run in P (or Q). Since every label on each transition is separate, $\hat{P}$ and $\hat{Q}$ are deterministic automata. Now, $\hat{P} \times \hat{Q}$ is constructed by ensuring that two transitions are combined in the product only if their alphabet is the same. Therefore if $(w, wt_1, l_1, wt_2, l_2) \in \hat{P} \times \hat{Q}$, then $\hat{\rho} = (w, wt_1, l_1) \in \hat{P}$, $\hat{\sigma} = (w, wt_2, l_2) \in \hat{Q}$. Hence, there exist runs ρ and σ with weight sequences wt_1 and wt_2 in P and Q, respectively. Next, $\hat{P} \times \hat{Q}$ is intersected over the weight sequences with ω-regular comparator $\mathcal{A}_f^{\leq}$ for aggregate function f and relation $\leq$. Therefore $(w, wt_1, l_1, wt_2, l_2) \in \mathit{DomProof}$ iff $f(wt_1) \leq f(wt_2)$. Therefore runs ρ in P and σ in Q are runs on the same word s.t. aggregate weight in P is less than or equal to that of σ in Q. Therefore Dom constitutes of these $\hat{\rho}$. Therefore Dom consists of $\hat{\rho}$ only if ρ is a dominated run in P w.r.t $P \subseteq_f Q$.

Every step of the algorithm has a two-way implication, hence it is also true that every dominated run in P w.r.t $P \subseteq_f Q$ is present in Dom. ■

Lemma 3.3 *Given weighted ω-automata P and Q and their ω-regular comparator $\mathcal{A}_f^{\leq}$ for aggregate function f and relation $\leq$.* InclusionRegular$(P, Q, \mathcal{A}_f)$ *returns* True *iff $P \subseteq_f Q$.*

Proof 5 $\hat{P}$ consists of all runs of P. Dom consists of all dominated run in P w.r.t $P \subseteq_f Q$. $P \subseteq_f Q$ iff every run of P is dominated w.r.t $P \subseteq_f Q$. Therefore $P \subseteq_f Q$ is given by whether $\hat{P} \equiv \mathit{Dom}$, where $\equiv$ denotes qualitative equivalence. ■

It is worth noting that Algorithm InclusionRegular can be easily adapted to solve *strict* quantitative inclusion, denoted $P \subset_f Q$, by repeating the same procedure with

the ω-regular comparator with the inequality relation $<$. In this case, a run ρ_P in P on word $w \in \Sigma^\omega$ is said to be *dominated* w.r.t $P \subset_f Q$ if there exists a run ρ_Q in Q on the same word w such that $wt_P(\rho_P) < wt_Q(\rho_Q)$. A similar adaption will work for quantitative equivalence, denoted $P \equiv_f Q$.

Lastly, we present the complexity analysis InclusionRegular:

Theorem 3.3 *Let P and Q be weighted ω-automata and $\mathcal{A}_f$ be an ω-regular comparator. Quantitative inclusion problem, quantitative strict-inclusion problem, and quantitative equivalence problem for ω-regular aggregate function f is* PSPACE*-complete.*

Proof 6 All operations in InclusionRegular until Line 4 are polytime operations in the size of weighted ω-automata P, Q and comparator $\mathcal{A}_f$. Hence, *Dom* is polynomial in size of P, Q and $\mathcal{A}_f$. Line 5 solves a PSPACE-complete problem. Therefore, the quantitative inclusion for ω-regular aggregate function f is in PSPACE in size of the inputs P, Q, and $\mathcal{A}_f$.

The PSPACE-hardness of the quantitative inclusion is established via reduction from the *qualitative* inclusion problem, which is PSPACE-complete. The formal reduction is as follows: Let P and Q be Büchi automata (with all states as accepting states). Reduce P, Q to weighted automata $\overline{P}$, $\overline{Q}$ by assigning a weight of 1 to each transition. Since all runs in $\overline{P}$, $\overline{Q}$ have the same weight sequence, weight of all words in $\overline{P}$ and $\overline{Q}$ is the same for any function f. It is easy to see $P \subseteq Q$ (qualitative inclusion) iff $\overline{P} \subseteq_f \overline{Q}$ (quantitative inclusion). ■

Theorem 3.3 extends to weighted ω-automata when weight of words is the *infimum* of weight of runs. The key idea for $P \subseteq_f Q$ here is to ensure that for every run ρ_Q in Q there exists a run on the same word in ρ_P in P s.t. $f(\rho_P) \leq f(\rho_Q)$.

Representation of counterexamples

When $P \not\subseteq_f Q$, there exists word(s) $w \in \Sigma^*$ s.t $wt_P(w) > wt_Q(w)$. Such a word w is said to be a *counterexample word*. The ability to extract counterexamples and analyze them has been of immense advantage in verification and synthesis in qualitative systems [21].

Here, we show how InclusionRegular can be adapted to yields Büchi automaton-representations for all counterexamples of an instance of quantitative inclusion for ω-regular comparators. Hopefully, this could be a starting point for counter-example guided frameworks in the quantitative verification of systems:

Theorem 3.4 *All counterexamples of the quantitative inclusion problem for an ω-regular aggregate function can be expressed by a Büchi automaton.*

Proof 7 For word w to be a counterexample, it must contain a run in P that is not dominated. Clearly, all non-dominated runs of P w.r.t to the quantitative inclusion are members of $\hat{P} \setminus Dom$. The counterexamples words can be obtained from $\hat{P} \setminus Dom$ by modifying its alphabet to the alphabet of P by dropping transition weights and their unique labels. ■

3.2.2 Quantitative games with perfect information

Recall, a *quantitative graph with complete information game*, referred to as graph game in short, is defined over a structure $G = (V = V_0 \uplus V_1, v_{init}, E, \gamma, f)$. It consists of a directed graph (V, E), and a partition (V_0, V_1) of its set of states V. State v_{init} is the *initial state* of the game. vE designates the set $\{w \in V | (v, w) \in E\}$ to indicate the successor states of state $v \in V$. For convenience, we assume that every state has at least one outgoing edge, i.e, $vE \neq \emptyset$ for all $v \in V$. Each transition of the

game is associated with a *cost* determined by the *cost function* $\gamma : E \to \mathbb{Z}$. Finally, $f : \mathbb{Z}^\omega \to \mathbb{R}$ is the *aggregate function*.

We illustrate how to solve the satisficing problem over quantitative games when the comparator automata for the aggregate function is ω-regular.

Wlog, let the winning condition for the maximizing player be to ensure that the cost of resulting play is greater than or equal to a given threshold value v. Then the winning condition for the minimizing player is to ensure that the cost of plays is less than threshold v.

For sake of simplicity, let the threshold value be 0. Further, suppose that $f(B) = 0$ where B is the sequence of all 0s. The we claim that the satisficing problem will reduce to solving a parity game.

Let $\mathcal{A}$ be the comparator automata for the inequality $\geq$. Currently $\mathcal{A}$ accepts a sequence (A, B) iff $f(A) \geq f(B)$. We can convert $\mathcal{A}$ to an NBA that accepts sequence A iff $f(A) \geq 0$ if we fix sequence B to 0^ω. Let us denote this by $\mathcal{A}_0$. Note that the size of $\mathcal{A}_0$ is the same as $\mathcal{A}$. Finally, determinize $\mathcal{A}_0$ into a parity automata P. Now, the winning condition of the maximizing agent in this game is that the weight sequence of a play must be accepted by the automaton P. Therefore, we take synchronized product of the quantitative game with the parity automata to obtain a parity game. Then the maximizing player will win the quantitative game iff it wins in the aforementioned parity game.

Theorem 3.5 *Given a quantitative game with complete information $\mathcal{G}$ and ω-regular comparator $\mathcal{A}_f$ for the aggregate function f. Let the threshold value be 0 and $f(0^\omega) = 0$. Then determining whether the maximizing agent has a winning strategy in the satisficing game with threshold 0 reduces to solving a parity game with $\mathcal{O}(|\mathcal{G}| \times |D_f|)$ states with $|\mathcal{A}_f|$ colors, where $|D| = |\mathcal{A}_f|^{|\mathcal{A}_f|}$.*

Proof 8 Since, D_f is the deterministic parity automaton equivalent to $A_f, |D_f| = |\mathcal{A}_f|^{O(|\mathcal{A}_f|)}$. The parity automaton will have $|\mathcal{A}_f|$ colors. Therefore, the product game will be a parity automaton with states equal to the product of the size of the game and the parity automaton, and colors equal to the size of the initial NBA. ■

3.2.3 Quantitative games with imperfect information

Given an incomplete-information quantitative game $\mathcal{G} = (S, s_{\mathcal{I}}, O, \Sigma, \delta, \gamma, f)$, our objective is to determine if player P_0 has a winning strategy $\alpha : O^* \to \Sigma$ for ω-regular aggregate function f. We assume we are given the ω-regular comparator $\mathcal{A}_f$ for function f. Note that a function $A^* \to B$ can be treated like a B-labeled A-tree, and vice-versa. Hence, we proceed by finding a Σ-labeled O-tree – the *winning strategy tree*. Every branch of a winning strategy-tree is an observed play o_ρ of $\mathcal{G}$ for which every actual play ρ is a winning play for P_0.

We first consider all *game trees* of $\mathcal{G}$ by interpreting $\mathcal{G}$ as a tree-automaton over Σ-labeled S-trees. Nodes $n \in S^*$ of the game-tree correspond to states in S and labeled by actions in Σ taken by player P_0. Thus, the *root node* ε corresponds to $s_{\mathcal{I}}$, and a node $s_{i_0}, \ldots, s_{i_k}$ corresponds to the state s_{i_k} reached via $s_{\mathcal{I}}, s_{i_0}, \ldots, s_{i_{k-1}}$. Consider now a node x corresponding to state s and labeled by an action σ. Then x has children $xs_1, \ldots xs_n$, for every $s_i \in S$. If $s_i \in \delta(s, \sigma)$, then we call xs_i a *valid* child, otherwise we call it an *invalid* child. Branches that contain invalid children correspond to invalid plays.

A game-tree τ is a *winning tree* for player P_0 if every branch of τ is either a winning play for P_0 or an invalid play of $\mathcal{G}$. One can check, using an automata, if a play is invalid by the presence of invalid children. Furthermore, the winning condition for P_0 can be expressed by the ω-regular comparator $\mathcal{A}_f$ that accepts (A, B)

iff $f(A) > f(B)$. To use the comparator $\mathcal{A}_f$, it is determinized to parity automaton D_f. Thus, a product of game $\mathcal{G}$ with D_f is a deterministic parity tree-automaton accepting precisely winning-trees for player P_0.

Winning trees for player P_0 are Σ-labeled S-trees. We need to convert them to Σ-labeled O-trees. Recall that every state has a unique observation. We can simulate these Σ-labeled S-trees on strategy trees using the technique of *thinning* states S to observations O [65]. The resulting alternating parity tree automaton $\mathcal{M}$ will accept a Σ-labeled O-tree τ_o iff for all actual game-tree τ of τ_o, τ is a winning-tree for P_0 with respect to the strategy τ_o. The problem of existence of winning-strategy for P_0 is then reduced to non-emptiness checking of $\mathcal{M}$.

Theorem 3.6 *Given an incomplete-information quantitative game $\mathcal{G}$ and ω-regular comparator $\mathcal{A}_f$ for the aggregate function f, the complexity of determining whether P_0 has a winning strategy is exponential in $|\mathcal{G}| \cdot |D_f|$, where $|D_f| = |\mathcal{A}_f|^{O(|\mathcal{A}_f|)}$.*

Proof 9 Since, D_f is the deterministic parity automaton equivalent to $A_f, |D_f| = |\mathcal{A}_f|^{O(|\mathcal{A}_f|)}$. The thinning operation is linear in size of $|\mathcal{G} \times D_f|$, therefore $|\mathcal{M}| = |\mathcal{G}| \cdot |D_f|$. Non-emptiness checking of alternating parity tree automata is exponential. Therefore, our procedure is doubly exponential in size of the comparator and exponential in size of the game. ■

The question of tighter bounds is open.

3.3 Chapter summary

This chapter identified a novel mode for comparison in quantitative systems: the online comparison of aggregate values of sequences of quantitative weights. This notion is embodied by comparators automata that read two infinite sequences of

weights synchronously and relate their aggregate values. We showed that ω-regular comparators not only yield generic algorithms for problems including quantitative inclusion and winning strategies in incomplete-information quantitative games, they also result in algorithmic advances.

We believe comparators, especially ω-regular comparators, can be of significant utility in verification and synthesis of quantitative systems, as demonstrated by the existence of finite-representation of counterexamples of the quantitative inclusion problem. Another potential application is computing equilibria in quantitative games. Applications of the prefix-average comparator, in general ω-context-free comparators, is open to further investigation. Another direction to pursue is to study aggregate functions in more detail, and attempt to solve the conjecture relating ω-regular aggregate functions and ω-regular comparators.

Chapter 4

On comparators of popular aggregate functions

Chapter 3 laid down the theoretical foundations of comparator automata, and demonstrated that for the class of aggregate functions that permit ω-regular comparators, comparator automata result in algorithms that are generalizable as well as integrated in approach. This chapter asks a concrete question: *Which aggregate functions allow ω-regular comparators?*

We know from Theorem 3.2 that limsup and liminf posses ω-regular comparators. This chapter investigates the comparators for commonly appearing aggregate functions, namely discounted-sum and limit-average. We observe that discounted-sum permits ω-regular comparators iff the discount factor is an integer (Section 4.1- 4.2), while limit-average does not permit ω-regular comparators (Section 4.3). In an attempt to establish a broader characterization of which aggregate function permit ω-regular comparators, we investigate the class of ω-regular functions. We deliver positive news: All ω-regular functions will have an ω-regular comparator (Section 4.4).

Last but not the least, despite being a generalizable framework, comparator-based algorithms may be better in complexity that existing approaches. In particular, for the case of DS with integer discount factors, we observe that the comparator-based algorithm is PSPACE as opposed to the previously known EXPTIME and EXPSPACE algorithm. This also establishes that DS inclusion for integer discount factor is PSPACE-complete, resolve the open question about its complexity class (Section 4.2.1).

4.1 Discounted-sum with non-integer discount factors

Recall, discounted-sum aggregation accumulates diminishing returns. This section establishes is that a DS comparison language and DS comparator are not ω-regular when the associated discount factor is not an integer.

We begin by formally defining DS comparison languages and DS comparator automata. Given an upper bound $\mu > 0$, discount-factor $d > 1$, and an inequality or equality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, the *DS comparison language with μ, d, and* R is a language over the alphabet $\Sigma \times \Sigma$, where $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu\}$, that accepts the pair of infinite-length weight sequences (A, B) iff $DS(A, d)$ R $DS(B, d)$ holds. Given an upper bound $\mu > 0$, discount-factor $d > 1$ and inequality or equality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, the *DS comparator automata (DS comparator, in short) for μ, d, and* R is the automaton that accepts the DS comparison language for μ, d, and R.

The proof begins with defining a *cut-point language* [38]: For a weighted ω-automaton $\mathcal{A}$ and a real number $r \in \mathbb{R}$, the *cut-point language* of $\mathcal{A}$ w.r.t. r is defined as $L^{\geq r} = \{w \in L(\mathcal{A}) | wt_{\mathcal{A}}(w) \geq r\}$. When the discount factor is a rational value $1 < d < 2$, it is known that not all deterministic weighted ω-automaton with discounted-sum aggregate function (DS-automaton, in short) have an ω-regular cut-point language for an $r \in \mathbb{R}$ [38]. In this section, this prior result is first extended to all non-integer, rational discount factors $d > 1$. Finally, this extension is utilized to establish that the DS comparison language for a non-integer, rational discount-factors $d > 1$ is not ω-regular. This proves that the DS aggregation function is not ω-regular when the discount factor is a non-integer, rational number.

Theorem 4.1 *Let $d > 1$ be a non-integer, rational discount factor. There exists a*

deterministic discounted-sum automata $\mathcal{A}$ and a rational value $r \in \mathbb{Q}$ such that its cut-point language w.r.t. r is not ω-regular.

Proof 10 Since the proof for $1 < d < 2$ has been presented in [38], we skip that case.

The proof presented here extends the earlier result on $1 < d < 2$ from [38] to all non-integer, rational discount factors $d > 1$.

Let $d > 2$ be a non-integer, rational discount-factor. Define deterministic discounted-sum automata $\mathcal{A}$ over the alphabet $\{0, 1, \ldots, \lceil d \rceil - 1\}$ such that the weight of transitions on alphabet $n \in \{0, 1, \ldots, \lceil d \rceil - 1\}$ is n. Therefore, weight of word $w \in \mathcal{A}$ is $DS(w, d)$. Consider its cut-point language $L^{\geq(\lceil d \rceil - 1)}$. We say a finite-length word w is *ambiguous* iff $\lceil d \rceil - 1 - \frac{1}{d^{|w|-1}} \cdot \frac{\lceil d \rceil - 1}{d-1} \leq DS(w, d) < \lceil d \rceil - 1$. Intuitively, a finite word w is ambiguous if it can be extended into infinite words in $\mathcal{A}$ such that some extensions lie in $L^{\geq(\lceil d \rceil - 1)}$ and some do not. Clearly, the finite word $w = \lceil d \rceil - 2$ is ambiguous. Note that when $d > 2$ is non-integer, rational valued, then $\frac{\lceil d \rceil - 1}{d-1} > 1$, so word $\lceil d \rceil - 2$ falls within the range for ambiguity. We claim that if finite word w is ambiguous, then either $w \cdot (\lceil d \rceil - 2)$ or $w \cdot (\lceil d \rceil - 1)$ is ambiguous. To prove ambiguity, we need to show that $\lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot \frac{\lceil d \rceil - 1}{d-1} \leq DS(w \cdot (\lceil d \rceil - k), d) < \lceil d \rceil - 1$, for $k \in \{1, 2\}$. Note, $DS(w \cdot (\lceil d \rceil - 2), d) = DS(w, d) + \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 2)$, and $DS(w \cdot (\lceil d \rceil - 1), d) = DS(w, d) + \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 1)$. Simplifying the expressions for $w \cdot (\lceil d \rceil - 1)$ and $w \cdot (\lceil d \rceil - 2)$, we need to prove either $\lceil d \rceil - 1 - \frac{1}{d^{|w|+1}} \cdot \frac{\lceil d \rceil - 1}{d-1} \leq DS(w, d) < \lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 1)$ or $\lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot \frac{\lceil d \rceil - 1}{d-1} - \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 2) \leq DS(w, d) < \lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 2)$. Now, this is true if $\lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot \frac{\lceil d \rceil - 1}{d-1} - \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 2) < \lceil d \rceil - 1 - \frac{1}{d^{|w|}} \cdot (\lceil d \rceil - 1)$ which is equivalent $d > 2$ is a is non-integer, rational discount-factor. Therefore, every ambiguous finite word can be extended to another ambiguous word. This means there exists an infinite word $w^{\geq}$ such that $DS(w^{\geq}, d) = \lceil d \rceil - 1$ and all finite prefixes of $w^{\geq}$ are ambiguous.

Let us assume that the language $L^{\geq(\lceil d \rceil - 1)}$ is ω-regular and represented

by Büchi automaton $\mathcal{B}$. For $n < m$, let the n- and m-length prefixes of $w^{\geq}$, denoted $w^{\geq}[0, n-1]$ and $w^{\geq}[0, m-1]$, respectively, be such that they reach the same states in $\mathcal{B}$. Then there exists an infinite length word w_s such that $DS(w^{\geq}[0, n-1] \cdot w_s, d) = DS(w^{\geq}[0, m-1] \cdot w_s, d) = \lceil d \rceil - 1$. Now, $DS(w^{\geq}[0, n-1] \cdot w_s, d) = DS(w^{\geq}[0, n-1], d) + \frac{1}{d^n} \cdot DS(w_s, d)$ and $DS(w^{\geq}[0, m-1] \cdot w_s, d) = DS(w^{\geq}[0, m-1], d) + \frac{1}{d^m} \cdot DS(w_s, d)$. Eliminating $DS(w_s, d)$ from the equations and simplification, we get:

$$d^{m-1} \cdot (DS(w^{\geq}[0, m-1], d) - (\lceil d \rceil - 1)) + d^{n-1} \cdot (DS(w^{\geq}[0, n-1], d) - (\lceil d \rceil - 1)) = 0$$

The above is a polynomial over d with degree $m-1$ and integer coefficients. Specifically, $d = \frac{p}{q} > 2$ such that integers $p, q > 1$, and p and q are mutually prime. Since $d = \frac{p}{q}$ is a root of the above equation, q must divide co-efficient of the highest degree term, in this case it is $m-1$. The co-efficient of the highest degree term in the polynomial above is $(w^{\geq}[0] - (\lceil d \rceil - 1))$. Recall from construction of $w^{\geq}$ above, $w^{\geq}[0] = \lceil d \rceil - 2$. So the co-efficient of the highest degree term is -1, which is not divisible by integer $q > 1$. Hence, resulting in a contradiction. ■

Finally, we use Theorem 4.1 to prove the DS comparison language is not ω-regular when the discount-factor $d > 1$ is not an integer.

Theorem 4.2 *Comparison language for the DS aggregation function is not ω-regular when the discount-factor $d > 1$ is a non-integer, rational number.*

Proof 11 To prove that the DS aggregation function is not ω-regular when the discount factor is not an integer, it is sufficient to prove that DS comparison language for $\geq$ is not ω-regular.

Let $d > 1$ be a non-integer, rational discount factor. Let $\mathcal{A}$ be the weighted ω-automaton as described in proof of Lemma 4.1. Consider its cut-point language

$L^{\geq(\lceil d\rceil-1)}$. From Lemma 4.1 and [38], we know that $L^{\geq(\lceil d\rceil-1)}$ is not an ω-regular language.

Suppose there exists an ω-regular DS comparator $\mathcal{A}_d^{\leq}$ for non-integer rational discount factor $d > 1$ for relation $\geq$. We define the Büchi automaton $\mathcal{P}$ s.t. $\mathcal{L}(\mathcal{P}) = \{(w, v) | w \in \mathcal{L}(\mathcal{A}), v = \lceil d\rceil - 1 \cdot 0^{\omega}\}$. Note that $DS(\lceil d\rceil - 1 \cdot 0^{\omega}, d) = \lceil d\rceil - 1$. . Then the cut-point language $L^{\geq(\lceil d\rceil-1)}$ of deterministic discounted-sum automata $\mathcal{A}$ can be constructed by taking the intersection of $\mathcal{P}$ with $\mathcal{A}_d^{\geq}$. Since all actions are closed under ω-regular operations, $L^{\geq 1}$ can be represented by a Büchi automaton. But this contradicts Theorem 4.1. Hence, our assumption cannot hold. ■

Since the DS comparison language with a non-integer discount-factor for any one inequality relation is not ω-regular, due to closure properties of ω-regular (Theorem 3.1-Item 3) this implies that the DS comparison language for all other inequities is also not ω-regular all inequalities is also not ω-regular. Finally, the cut-point argument can be extended to $=$ and $\neq$ relation to show that their DS comparison languages with non-integer discount-factors is also not ω-regular.

4.2 Discounted-sum with integer discount factors

This section proves that a DS comparison languages is ω-regular when the discount-factor $d > 1$ is an integer. Hence, DS aggregation function is ω-regular for integer discount-factors. The result is proven by explicitly constructing a Büchi automaton that accepts the DS comparison language for an arbitrary upper bound μ, an inequality relation R and integer discount factor $d > 1$.

An immediate side-effect of this result is that DS inclusion is PSPACE-complete. Not only does this improve upon the previously best known algorithm that was

EXPTIME and EXPSPACE, it also resolves 15-year long open question of the complexity class of DS inclusion with integer discount factor (Section 4.2.1).

Core intuition

Recall the definitions of DS comparison language and DS comparator automata from Section 4.1.

Let integer $\mu > 0$ be the upper-bound on sequences. The core intuition is that bounded sequences can be converted to their value in an integer base d via a finite-state transducer. Lexicographic comparison of the converted sequences renders the desired DS-comparator. Conversion of sequences to base d requires a certain amount of *look-ahead* by the transducer. Here we describe a method that directly incorporates the look-ahead with lexicographic comparison to obtain the Büchi automaton corresponding to the DS comparator for integer discount factor $d > 1$.

Construction details

We explain the construction in detail now. We complete the construction for the relation $<$. For sake of simplicity, we assume that sequences the weight sequences are *positive* integer sequences. Under this assumption, for a weight sequence A and integer discount-factor $d > 1$, $DS(A, d)$ can be interpreted as a value in base d i.e. $DS(A, d) = A[0] + \frac{A[1]}{d} + \frac{A[2]}{d^2} + \cdots = (A[0].A[1]A[2]\dots)_d$ [42]. The proofs can be extended to integer sequences easily.

Unlike comparison of numbers in base d, the lexicographically larger sequence may not be larger in value since (i) The elements of weight sequences may be larger in value than base d, and (ii) Every value has multiple infinite-sequence representations. To overcome the two challenges mentioned above, we resort to arithmetic techniques

in base d. Note that $DS(B,d) > DS(A,d)$ iff there exists a sequence C such that $DS(B,d) = DS(A,d) + DS(C,d)$, and $DS(C,d) > 0$. Therefore, to compare the discounted-sum of A and B, the objective is to obtain the sequence C. Arithmetic in base d also results in sequence X of carry elements. Then:

Lemma 4.1 *Let A, B, C, X be weight sequences, $d > 1$ be a positive integer such that following equations holds true:*

1. *When $i = 0$, $A[0] + C[0] + X[0] = B[0]$*
2. *When $i \geq 1$, $A[i] + C[i] + X[i] = B[i] + d \cdot X[i-1]$*

Then $DS(B,d) = DS(A,d) + DS(C,d)$.

Proof 12 $DS(A,d) + DS(C,d) = \Sigma_{i=0}^{\infty} A[i]\frac{1}{d^i} + \Sigma_{i=0}^{\infty} C[i]\frac{1}{d^i} = \Sigma_{i=0}^{\infty}(A[i] + C[i])\frac{1}{d^i} = (B[0] - X[0]) + \Sigma_{i=1}^{\infty}(B[i] + d \cdot X[i-1] - X[i])\frac{1}{d^i} = (B[0] - X[0]) + \Sigma_{i=1}^{\infty}(B[i] + d \cdot X[i-1] - X[i])\frac{1}{d^i} = \Sigma_{i=0}^{\infty} B[i] \cdot \frac{1}{d^i} - \Sigma_{i=0}^{\infty} X[i] + \Sigma_{i=0}^{\infty} X[i] = \Sigma_{i=0}^{\infty} B[i] \cdot \frac{1}{d^i} = DS(B,d)$ ■

Hence to determine $DS(B,d) - DS(A,d)$, systematically guess sequences C and X using the equations, element-by-element beginning with the 0-th index and moving rightwards. There are two crucial observations here: (i) Computation of i-th element of C and X only depends on i-th and $(i-1)$-th elements of A and B. Therefore guessing $C[i]$ and $X[i]$ requires *finite memory* only. (ii) Intuitively, C refers to a representation of value $DS(B,d) - DS(A,d)$ in base d and X is the carry-sequence. If we can prove that X and C are also bounded-sequences and can be constructed from a finite-set of integers, we would be able to further proceed to construct a Büchi automaton for the desired comparator.

We proceed by providing an inductive construction of sequences C and X that satisfy properties in Lemma 4.1, and show that these sequences are bounded when A

and B are bounded. In particular, when A and B are bounded integer-sequences, then sequences C and X constructed here are also bounded-integer sequences. Therefore, they are be constructed from a finite-set of integers. Proofs for sequence C are in Lemma 4.3-Lemma 4.5, and proof for sequence X is in Lemma 4.6.

We begin with introducing some notation. Let $DS^{-}(B, A, d, i) = \Sigma_{j=0}^{i}(B[j] - A[j]) \cdot \frac{1}{d^j}$ for all index $i \geq 0$. Also, let $DS^{-}(B, A, d, \cdot) = \Sigma_{j=0}^{\infty}(B[j] - A[j]) \cdot \frac{1}{d^j} = DS(B, d) - DS(A, d)$. Define $maxC = \mu \cdot \frac{d}{d-1}$. We define the residual function $Res : \mathbb{N} \cup \{0\} \mapsto \mathbb{R}$ as follows:

$$Res(i) = \begin{cases} DS^{-}(B, A, d, \cdot) - \lfloor DS^{-}(B, A, d, \cdot) \rfloor & \text{if } i = 0 \\ Res(i-1) - \lfloor Res(i-1) \cdot d^i \rfloor \cdot \frac{1}{d^i} & \text{otherwise} \end{cases}$$

Then we define $C[i]$ as follows:

$$C[i] = \begin{cases} \lfloor DS^{-}(B, A, d, \cdot) \rfloor & \text{if } i = 0 \\ \lfloor Res(i-1) \cdot d^i \rfloor & \text{otherwise} \end{cases}$$

Intuitively, $C[i]$ is computed by *stripping off* the value of the i-th digit in a representation of $DS^{-}(B, A, d, \cdot)$ in base d. $C[i]$ denotes the numerical value of the i-th position of the difference between B and A. The residual function denotes the numerical value of the difference remaining after assigning the value of $C[i]$ until that i.

We define function $CSum(i) : \mathbb{N} \cup \{0\} \to \mathbb{Z}$ s.t. $CSum(i) = \Sigma_{j=0}^{i} C[j] \cdot \frac{1}{d^j}$. Then, we define $X[i]$ as follows:

$$X[i] = (DS^{-}(B, A, d, i) - CSum(i)) \cdot d^i$$

Therefore, we have defined sequences C and X as above. We now prove the desired properties one-by-one.

First, we establish sequences C, X as defined here satisfy Equations 1-2 from Lemma 4.1. Therefore, ensuring that C is indeed the difference between sequences B and A, and X is their carry-sequence.

Lemma 4.2 *Let A and B be bounded integer sequences and C and X be defined as above. Then,*

1. $B[0] = A[0] + C[0] + X[0]$
2. *For $i \geq 1$,* $B[i] + d \cdot X[i-1] = A[i] + C[i] + X[i]$

Proof 13 We prove this by induction on i using definition of function X.

When $i = 0$, then $X[0] = DS^{-}(B, A, d, 0) - CSum(0) \implies X[0] = B[0] - A[0] - C[0] \implies B[0] = A[0] + C[0] + X[0]$.

When $i = 1$, then $X[1] = (DS^{-}(B, A, d, 1) - CSum(1)) \cdot d = (B[0] + B[1] \cdot \frac{1}{d}) - (A[0] + A[1] \cdot \frac{1}{d}) - (C[0] + C[1] \cdot \frac{1}{d})) \cdot d \implies X[1] = B[0] \cdot d + B[1] - (A[0] \cdot d + A[1]) - (C[0] \cdot d + C[1])$. From the above we obtain $X[1] = d \cdot X[0] + B[1] - A[1] - C[1] \implies B[1] + d \cdot X[0] = A[1] + C[1] + X[1]$.

Suppose the invariant holds true for all $i \leq n$, we show that it is true for $n + 1$. $X[n+1] = (DS^{-}(B, A, d, n+1) - CSum(n+1)) \cdot d^{n+1} \implies X[n+1] = (DS^{-}(B, A, d, n) - CSum(n)) \cdot d^{n+1} + (B[n+1] - A[n+1] - C[n+1]) \implies X[n+1] = X[n] \cdot d + B[n+1] - A[n+1] - C[n+1] \implies B[n+1] + X[n] \cdot d = A[n+1] + C[n+1] + X[n+1]$. ■

Next, we establish the sequence C is a bounded integer-sequences, therefore it can be represented by a finite-set of integers. First of all, by definition of $C[i]$ it is clear that $C[i]$ is an integer for all $i \geq 0$. We are left with proving boundedness of C. Lemma 4.3-Lemma 4.5 establish boundedness of $C[i]$.

Lemma 4.3 *For all $i \geq 0$, $Res(i) = DS^{-}(B, A, d, \cdot) - CSum(i)$.*

Proof 14 Proof by simple induction on the definitions of functions Res and C.

1. When $i = 0$, $Res(0) = DS^{-}(B, A, d, \cdot) - \lfloor DS^{-}(B, A, d, \cdot) \rfloor$. By definition of $C[0]$, $Res(0) = DS^{-}(B, A, d, \cdot) - C[0] \iff Res(0) = DS^{-}(B, A, d, \cdot) - CSum(0)$.

2. Suppose the induction hypothesis is true for all $i < n$. We prove it is true when $i = n$. When $i = n$, $Res(n) = Res(n-1) - \lfloor Res(n-1) \cdot d^{n} \rfloor \cdot \frac{1}{d^{n}}$. By definition of $C[n]$ and I.H, we get $Res(n) = (DS^{-}(B, A, d, \cdot) - CSum(n-1)) - C[n] \cdot \frac{1}{d^{n}}$. Therefore $Res(n) = DS^{-}(B, A, d, \cdot) - CSum(n)$.

■

Lemma 4.4 *When $DS^{-}(B, A, d, \cdot) \geq 0$, for all $i \geq 0$, $0 \leq Res(i) < \frac{1}{d^{i}}$.*

Proof 15 Since, $DS^{-}(B, A, d, \cdot) \geq 0$, $Res(0) = DS^{-}(B, A, d, \cdot) - \lfloor DS^{-}(B, A, d, \cdot) \rfloor \geq 0$ and $Res(0) = DS^{-}(B, A, d, \cdot) - \lfloor DS^{-}(B, A, d, \cdot) \rfloor < 1$. Specifically, $0 \leq Res(0) < 1$.

Suppose for all $i \leq k$, $0 \leq Res(i) < \frac{1}{d^{i}}$. We show this is true even for $k + 1$.

Since $Res(k) \geq 0$, $Res(k) \cdot d^{k+1} \geq 0$. Let $Res(k) \cdot d^{k+1} = x + f$, for integral $x \geq 0$, and fractional $0 \leq f < 1$. Then, from definition of Res, we get $Res(k+1) = \frac{x+f}{d^{k+1}} - \frac{x}{d^{k+1}} \implies Res(k+1) < \frac{1}{d^{k+1}}$.

Also, $Res(k+1) \geq 0$ since $a - \lfloor a \rfloor \geq 0$ for all positive values of a (Lemma 4.3). ■

Lemma 4.5 *Let $maxC = \mu \cdot \frac{d}{d-1}$. When $DS^{-}(B, A, d, \cdot) \geq 0$, for $i = 0$, $0 \leq C(0) \leq maxC$, and for $i \geq 1$, $0 \leq C(i) < d$.*

Proof 16 Since both A and B are non-negative bounded weight sequences, maximum value of $DS^{-}(B, A, d, \cdot)$ is when $B = \{\mu\}_i$ and $A = \{0\}_i$. In this case $DS^{-}(B, A, d, \cdot) = maxC$. Therefore, $0 \leq C[0] \leq maxC$.

From Lemma 4.4, we know that for all i, $0 \leq Res(i) < \frac{1}{d^i}$. Alternately, when $i \geq 1$, $0 \leq Res(i-1) < \frac{1}{d^{i-1}} \implies 0 \leq Res(i-1) \cdot d^i < \frac{1}{d^{i-1}} \cdot d^i \implies 0 \leq Res(i-1) \cdot d^i < d \implies 0 \leq \lfloor Res(i-1) \cdot d^i \rfloor < d \implies 0 \leq C[i] < d.$ ■

Therefore, we have established that sequence C is non-negative integer-valued and is bounded by $maxC = \mu \cdot \frac{d}{d-1}$.

Finally, we prove that sequence X is also a bounded-integer sequence, thereby proving that it is bounded, and can be represented with a finite-set of integers. Note that for all $i \geq 0$, by expanding out the definition of $X[i]$ we get that $X[i]$ is an integer for all $i \geq 0$. We are left with proving boundedness of X:

Lemma 4.6 *Let* $maxX = 1 + \frac{\mu}{d-1}$. *When* $DS^-(B, A, d, \cdot) \geq 0$, *then for all* $i \geq 0$, $|X(i)| \leq maxX$.

Proof 17 From definition of X, we know that $X(i) = (DS^-(B, A, d, i) - CSum(i)) \cdot d^i \implies X(i) \cdot \frac{1}{d^i} = DS^-(B, A, d, i) - CSum(i)$. From Lemma 4.3 we get $X(i) \cdot \frac{1}{d^i} = DS^-(B, A, d, i) - (DS^-(B, A, d, \cdot) - Res(i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (DS^-(B, A, d, \cdot) - DS^-(B, A, d, i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (\Sigma_{j=i+1}^{\infty}(B[j] - A[j]) \cdot \frac{1}{d^j}) \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + |(\Sigma_{j=i+1}^{\infty}(B[j] - A[j]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |(\Sigma_{j=0}^{\infty}(B[j + i + 1] - A[j + i + 1]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |maxC|$. From Lemma 4.4, this implies $|X(i) \cdot \frac{1}{d^i}| \leq \frac{1}{d^i} + \frac{1}{d^{i+1}} \cdot |maxC| \implies |X(i)| \leq 1 + \frac{1}{d} \cdot |maxC| \implies |X(i)| \leq 1 + \frac{\mu}{d-1} \implies |X(i)| \leq maxX.$ ■

We summarize our results from 4.2-Lemma 4.6 as follows:

Corollary 4.1 *Let* $d > 1$ *be an integer discount-factor. Let* A *and* B *be non-negative integer sequences bounded by* μ, *and* $DS(A, d) < DS(B, d)$. *Then there exists bounded*

integer-valued sequences X and C that satisfy the conditions in Lemma 4.1. Furthermore, C and X are bounded as follows:

1. $0 \leq C[0] \leq \mu \cdot \frac{d}{d-1}$ *and for all* $i \geq 1$, $0 \leq C[i] < d$,
2. *For all* $i \geq 0$, $0 \leq |X[i]| \leq 1 + \frac{\mu}{d-1}$

Intuitively, we construct a Büchi automaton $\mathcal{A}_d^<$ with states of the form (x, c) where x and c range over all possible values of X and C, respectively, and a special initial state s. Transitions over alphabet (a, b) replicate the equations in Lemma 4.1. i.e. transitions from the start state $(s, (a, b), (x, c))$ satisfy $a+c+x = b$ to replicate Equation 1 (Lemma 4.1) at the 0-th index, and all other transitions $((x_1, c_1), (a, b), (x_2, c_2))$ satisfy $a + c_2 + x_2 = b + d \cdot x_1$ to replicate Equation 2 (Lemma 4.1) at indexes $i > 0$. The complete construction is as follows:

Full and final construction Let $d > 1$ be an integer discount factor, $\mu > 0$ be the upper bound, and $<$ be the strict inequality relation. The DS comparator automata with discount factor $d > 1$, upper bound μ, and relation $<$ is constructed as follows:

Let $\mu_C = \mu \cdot \frac{d}{d-1}$ and $\mu_X = 1 + \frac{\mu}{d-1}$. Then, construct Büchi automaton $\mathcal{A}_d^< = (S, \Sigma, \delta_d, \mathit{Init}, \mathcal{F})$ such that,

- $S = \{s\} \cup \mathcal{F} \cup S_\perp$ where

  $\mathcal{F} = \{(x, c) || x| \leq \mu_X, 0 \leq c \leq \mu_C\}$, and

  $S_\perp = \{(x, \perp) || x| \leq \mu_X\}$ where $\perp$ is a special character, and $c \in \mathbb{N}$, $x \in \mathbb{Z}$.

- State s is the initial state, and $\mathcal{F}$ are accepting states

- $\Sigma = \{(a, b) : 0 \leq a, b \leq \mu\}$ where a and b are integers.

- $\delta_d \subseteq S \times \Sigma \times S$ is defined as follows:

1. Transitions from start state s:

   i $(s, (a, b), (x, c))$ for all $(x, c) \in \mathcal{F}$ s.t. $a + x + c = b$ and $c \neq 0$

   ii $(s, (a, b), (x, \perp))$ for all $(x, \perp) \in S_\perp$ s.t. $a + x = b$

2. Transitions within $S_\perp$: $((x, \perp), (a, b), (x', \perp))$ for all $(x, \perp)$, $(x', \perp) \in S_\perp$, if $a + x' = b + d \cdot x$

3. Transitions within $\mathcal{F}$: $((x, c), (a, b), (x', c'))$ for all (x, c), $(x', c') \in \mathcal{F}$ where $c' < d$, if $a + x' + c' = b + d \cdot x$

4. Transition between $S_\perp$ and $\mathcal{F}$: $((x, \perp), (a, b), (x', c'))$ for all $(x, \perp) \in S_\perp$, $(x', c') \in \mathcal{F}$ where $0 < c' < d$, if $a + x' + c' = b + d \cdot x$

Theorem 4.3 *Let $d > 1$ be an integer discount-factor, and $\mu > 1$ be an integer upper-bound. Büchi automaton $\mathcal{A}_d^<$ represents the DS comparator automata for μ, d, and $<$. Büchi automaton $\mathcal{A}_d^<$ has $\mathcal{O}(\frac{\mu^2}{d})$-many states.*

Proof 18 Corollary 4.1 proves that if $DS(A, d) < DS(B, d)$ then sequence X and C satisfying the integer sequence criteria and bounded-criteria will exist. Let these sequences be $X = X[0]X[1]\ldots$ and $C = [0]C[1]\ldots$. Since $DS(C, d) > 0$, there exists an index $i \geq 0$ where $C[i] > 0$. Let the first position where $C[i] > 0$ be index j. By construction of $\mathcal{A}_d^<$, the state sequence given by $s, (X[0], \perp)\ldots, (X[j-1], \perp), (X[j], C[j]), (X[j+1], C[j+1])\ldots$, where for all $i \geq j$, $C[i] \neq \perp$, forms a run of word (A, B) in the Büchi automaton. Furthermore, this run is accepting since state (x, c) where $c \neq \perp$ are accepting states. Therefore, (A, B) is an accepting word in $\mathcal{A}_d^<$.

To prove the other direction, suppose the pair of sequence (A, B) has an accepting run with state sequence s, $(x_0, \perp), \ldots (x_{j-1}, \perp), (x_j, c_j), (x_{j+1}, c_{j+1})\ldots$, where for all

$i \geq j$, $c_j \neq \perp$. Construct sequence X and C as follows: For all $i \geq 0$, $X[i] = x_i$. For all $i < j$, $C[i] = 0$ and for all $i \geq j$ $C[i] = c_i$. Then the transitions of $\mathcal{A}_d^<$ guarantees equations Equation 1- 2 from Lemma 5.5 to hold for sequences A,B and C,X. Therefore, it must be the case that $DS(B, d) = DS(A, d) + DS(C, d)$. Furthermore, since the first transition to accepting states (x, c) where $c \neq \perp$ is possible only if $c > 0$, $DS(C, d) > 0$. Therefore, $DS(A, d) < DS(B, d)$. Therefore, $\mathcal{A}_d^<$ accepts (A, B) if $DS(A, d) < DS(B, d)$. ■

Corollary 4.2 *Comparison languages for DS aggregation function with an integer discount-factor $d > 1$ is ω-regular.*

Proof 19 Immediate from Theorem 4.3, and closure properties of ω-regular comparison languages (Theorem 3.1-Item-1). ■

Note that the proof of Theorem 3.1-Item-1 also gives a way to construct the DS comparators for all other equality or inequality relations. However, if those were to be followed, then the resulting DS comparators may have a very large number of states. For instance, the proof of of Theorem 3.1-Item-1 suggests to construct the DS comparator for $\geq$ by taking the complement of $\mathcal{A}_d^<$ constructed above. Büchi complementation [80] will result in an exponential blow-up in the state space. Hence, this method of constructing DS comparators for the remaining relations is not practical. The silver lining here is that the construction of $\mathcal{A}_d^<$ can be modified in trivial ways to obtain the DS comparators for all other relations. Hence,

Theorem 4.4 *Let $d > 1$ be an integer discount-factor, and $\mu > 1$ be an integer upper bound. The Büchi automaton for the DS comparator for μ, d, and R for $\mathsf{R} \in \{\leq, \geq, <, >, =, \neq\}$ consists of $\mathcal{O}(\frac{\mu^2}{d})$-many states.*

Proof 20 One could use the constructions from Theorem 3.1-Item-1 to obtain the DS comparators for all other inequalities and equalities. However, those constructions would lead to comparators with larger state space than $\mathcal{O}(\frac{\mu^2}{d})$. In this proof, we will illustrate how to construct DS comparators for all other inequality and equalities by modifying the DS comparator for $<$ from Theorem 4.2 such that the resulting comparator has the size size as $\mathcal{O}(\frac{\mu^2}{d})$.

For $>$: The DS comparator for $>$ can be obtained by flipping the order of alphabet on the transitions. Specifically, for every transition in DS comparator for $<$, if the alphabet is (a, b), then switch the alphabet to (b, a) in the DS comparator for $>$.

For $\leq$: The DS comparator for $\leq$ can be constructed from the DS comparator for $<$ by changing the accepting states only. For the DS comparator for $\leq$, let the accepting states be $\mathcal{F} \cup S_\perp$. Intuitively, the states $S_\perp$ are those where the discounted-sum of the difference sequence C is 0, since sequence $C = 0^\omega$. Therefore, by adding $S_\perp$ to the accepting states of DS comparator for $<$, we have included all those sequence for which $DS(C, d) = 0$. Thereby, converting it to the DS comparator for $\leq$.

For $=$: In this case, set the accepting states in the DS comparator for $<$ to $S_\perp$ only. This way, the automaton will accept a sequence (A, B) iff the corresponding C is such that $DS(C, d) = 0$ since $C = 0\omega$.

For $\geq$: As done for constructing the DS comapartor for $>$ from DS comparator for $<$, simply swap the alphabet in the DS comparator for $\leq$ to get the comparator for $\geq$.

For $\neq$: Intuitively, one would want to take the $S_\perp$ states, but make sure they are not accepting. Next, one would appropriately add the (x, c) states from the DS comparators for both $<$ and $>$ in order to accept sequences (A, B) iff either $DS(A, d) > DS(B, d)$ or $DS(A, d) < DS(B, d)$. ■

4.2.1 Complexity of DS inclusion with integer discount factors

Finally, we utilize the ω-regularity of DS comparators with integer discount factors to establish that DS inclusion is PSPACE-complete when the discount factor is an integer. The prior best known algorithm is EXPTIME and EXPSPACE [31, 39], which does not match with its known PSAPCE lower bound.

Theorem 4.5 *Let integer $\mu > 1$ be the maximum weight on transitions in DS-automata P and Q, and $d > 1$ be an integer discount-factor. Let μ and d be represented in unary form. Then DS-inclusion, DS-strict-inclusion, and DS-equivalence between are* PSPACE*-complete.*

Proof 21 Since size of DS-comparator is polynomial w.r.t. to upper bound μ, when represented in unary, (Theorem 4.3), DS-inclusion is PSPACE in size of input weighted ω-automata and μ (Theorem 3.3). ■

By closing the gap between upper and lower bounds for DS inclusion, Theorem 4.5 resolves a 15-year old open problem. The earlier known EXPTIME upper bound in complexity is based on an exponential determinization construction (subset construction) combined with arithmetical reasoning [31,39]. We observe that the determinization construction can be performed on-the-fly in PSPACE. To perform, however, the arithmetical reasoning on-the-fly in PSPACE would require essentially using the same bit-level ((x, c)-state) techniques that we have used to construct DS-comparator.

4.3 Limit-average aggregation function

The limit-average of a sequence refers to the point of convergence of the average of prefixes of the sequence. Unlike discounted-sum, limit average is known to not

converge for all sequences. To work around this limitation, most applications simply use limit-average infimum or limit-average supremum of sequences [33,39,40,93]. We argue that the usage of limit-average infimum or limit-average supremum in liu of limit-average for purpose of comparison can be misleading. For example, consider sequence A s.t. $\mathsf{LimAvgSup}(A) = 2$ and $\mathsf{LimAvgInf}(A) = 0$, and sequence B s.t. $\mathsf{LimAvg}(B) = 1$. Clearly, limit-average of A does not exist. So while it is true that$\mathsf{LimAvgInf}(A) < \mathsf{LimAvgInf}(B)$, indicating that at infinitely many indices the average of prefixes of A is lower, this renders an incomplete picture since at infinitely many indices, the average of prefixes of B is greater as $\mathsf{LimAvgSup}(A) = 2$.

Such inaccuracies in limit-average comparison may occur when the limit-average of at least one sequence does not exist. However, it is not easy to distinguish sequences for which limit-average exists from those for which it doesn't.

We define *prefix-average comparison* as a relaxation of limit-average comparison. Prefix-average comparison coincides with limit-average comparison when limit-average exists for both sequences. Otherwise, it determines whether eventually the average of prefixes of one sequence are greater than those of the other. This comparison does not require the limit-average to exist to return intuitive results. Further, we show that the *prefix-average comparator* is ω-context-free.

4.3.1 Limit-average language and comparison

Let $\Sigma = \{0, 1, \ldots, \mu\}$ be a finite alphabet with $\mu > 0$. The *limit-average language* $\mathcal{L}_{LA}$ contains the sequence (word) $A \in \Sigma^\omega$ iff its limit-average exists. Suppose $\mathcal{L}_{LA}$ were ω-regular, then $\mathcal{L}_{LA} = \bigcup_{i=0}^{n} U_i \cdot V_i^\omega$, where $U_i, V_i \subseteq \Sigma^*$ are regular languages over *finite* words. The limit-average of sequences is determined by its behavior in the limit, so limit-average of sequences in V_i^ω exists. Additionally, the average of

all (finite) words in V_i must be the same. If this were not the case, then two words in V_i with unequal averages l_1 and l_2, can generate a word $w \in V_i^\omega$ s.t the average of its prefixes oscillates between l_1 and l_2. This cannot occur, since limit-average of w exists. Let the average of sequences in V_i be a_i, then limit-average of sequences in V_i^ω and $U_i \cdot V_i^\omega$ is also a_i. This is contradictory since there are sequences with limit-average different from the a_i (see appendix). Similarly, since every ω-CFL is represented by $\bigcup_{i=1}^{n} U_i \cdot V_i^\omega$ for CFLs U_i, V_i over finite words [45], a similar argument proves that $\mathcal{L}_{LA}$ is not ω-context-free.

Theorem 4.6 *$\mathcal{L}_{LA}$ is neither an ω-regular nor an ω-context-free language.*

Proof 22 We first prove that $\mathcal{L}_{LA}$ is not ω-regular.

Let us assume that the language $\mathcal{L}_{LA}$ is ω-regular. Then there exists a finite number n s.t. $\mathcal{L}_{LA} = \bigcup_{i=0}^{n} U_i \cdot V_i^\omega$, where U_i and $V_i \in \Sigma^*$ are regular languages over finite words.

For all $i \in \{0, 1, \ldots n\}$, the limit-average of any word in $U_i \cdot V_i^\omega$ is given by the suffix of the word in V_i^ω. Since $U_i \cdot V_i^\omega \subseteq \mathcal{L}_{LA}$, limit-average exists for all words in $U_i \cdot V_i^\omega$. Therefore, limit-average of all words in V_i^ω must exist. Now as discussed above, the average of all words in V_i must be the same. Furthermore, the limit-average of all words in V_i^ω must be the same, say $\mathsf{LimAvg}(w) = a_i$ for all $w \in V_i^\omega$.

Then the limit-average of all words in $\mathcal{L}_{LA}$ is one of $a_0, a_1 \ldots a_n$. Let $a = \frac{p}{q}$ s.t $p < q$, snd $a \neq a_i$ for $i \in \{0, 1, \ldots, \mu\}$. Consider the word $w = (1^p 0^{q-p})^\omega$. It is easy to see the $\mathsf{LimAvg}(w) = a$. However, this word is not present in $\mathcal{L}_{LA}$ since the limit-average of all words in $\mathcal{L}_{LA}$ is equal to a_0 or $a_1 \ldots$ or a_n.

Therefore, our assumption that $\mathcal{L}_{LA}$ is an ω-regular language has been contradicted.

Next we prove that $\mathcal{L}_{LA}$ is not an ω-CFL.

Every ω-context-free language can be written in the form of $\bigcup_{i=0}^{n} U_i \cdot V_i^{\omega}$ where U_i and V_i are context-free languages over finite words. The rest of this proof is similar to the proof for non-ω-regularity of $\mathcal{L}_{LA}$. ■

In the next section, we will define *prefix-average comparison* as a relaxation of limit-average comparison. To show how prefix-average comparison relates to limit-average comparison, we will require the following two lemmas: Quantifiers $\exists^{\infty} i$ and $\exists^{f} i$ denote the existence of *infinitely* many and *only finitely* many indices i, respectively.

Lemma 4.7 *Let A and B be sequences s.t. their limit average exists. If $\exists^{\infty} i, \mathsf{Sum}(A[0, i-1]) \geq \mathsf{Sum}(B[0, i-1])$ then $\mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$.*

Proof 23 Let the limit average of sequence A, B be a, b respectively. Since the limit average of A and B exists, for every $\epsilon > 0$, there exists N_{ϵ} s.t. for all $n > N_{\epsilon}$, $|\mathsf{Avg}(A[0, n-1]) - a| < \epsilon$ and $|\mathsf{Avg}(B[0, n-1]) - b| < \epsilon$.

Let $a - b = k > 0$.

Take $\epsilon = \frac{k}{4}$. Then for all $n > N_{\frac{k}{4}}$, since $|\mathsf{Avg}(A[0, n-1]) - a| < \epsilon$, $|\mathsf{Avg}(B[0, n-1]) - b| < \epsilon$ and that $a - b = k > 0$, $\mathsf{Avg}(A[0, n-1]) - \mathsf{Avg}(B[0, n-1]) > \frac{k}{2} \implies \frac{\mathsf{Sum}(A[0,n-1])}{n} - \frac{\mathsf{Sum}(B[0,n-1])}{n} > \frac{k}{2} \implies \mathsf{Sum}(A[0, n-1]) - \mathsf{Sum}(B[0, n-1]) > 0$.

Specifically, $\exists^{\infty} i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$. Furthermore, since there is no index greater than $N_{\frac{k}{4}}$ where $\mathsf{Sum}(A[0, n-1]) \leq \mathsf{Sum}(B[0, n-1])$, $\exists^{f} i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$. ■

Lemma 4.8 *Let A, B be sequences s.t their limit-average exists. If $\mathsf{LimAvg}(A) > \mathsf{LimAvg}(B)$ then $\exists^{f} i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$ and $\exists^{\infty} i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$.*

Proof 24 Let the limit-average of sequence A, B be L_a, L_b respectively. Since, the limit average of both A and B exists, for every $\epsilon > 0$, there exists N_ϵ s.t. for all $n > N_\epsilon$, $|\mathsf{Avg}(A[1,n]) - L_a| < \epsilon$ and $|\mathsf{Avg}(B[1,n]) - L_b| < \epsilon$.

Suppose it were possible that $\mathsf{LimAvg}(A) < \mathsf{LimAvg}(B)$. Suppose $L_b - L_a = k > 0$. Let $\epsilon = \frac{k}{4}$. By arguing as in Lemma 4.7, it must be the case that for all $n > N_{\frac{k}{4}}$, $\mathsf{Sum}(B[1,n]) - \mathsf{Sum}(A[1,n]) > 0$. But this is not possible, since we are given that $\exists^\infty i, \mathsf{Sum}(A[0,i-1]) > \mathsf{Sum}(B[0,i-1])$ (or $\exists^\infty i, \mathsf{Sum}(A[0,i-1] \geq \mathsf{Sum}(B[0,i-1]))$). Hence $\mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$. ■

4.3.2 Prefix-average comparison and comparator

The previous section relates limit-average comparison with the sums of equal length prefixes of the sequences (Lemma 4.7-4.8). The comparison criteria is based on the number of times sum of prefix of one sequence is greater than the other, which does not rely on the existence of limit-average. Unfortunately, this criteria cannot be used for limit-average comparison since it is incomplete (Lemma 4.8). Specifically, for sequences A and B with equal limit-average it is possible that $\exists^\infty i, \mathsf{Sum}(A[0,n-1]) > \mathsf{Sum}(B[0,n-1])$ and $\exists^\infty i, \mathsf{Sum}(B[0,n-1]) > \mathsf{Sum}(A[0,n-1])$. Instead, we use this criteria to define *prefix-average comparison*. In this section, we define prefix-average comparison and explain how it relaxes limit-average comparison. Lastly, we construct the prefix-average comparator, and prove that it is not ω-regular but is ω-context-free.

Definition 4.1 (Prefix-average comparison for relation $\geq$) Let A and B be number sequences. We say $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$ if $\exists^f i, \mathsf{Sum}(B[0,i-1]) \geq \mathsf{Sum}(A[0,i-1])$ and $\exists^\infty i, \mathsf{Sum}(A[0,i-1]) > \mathsf{Sum}(B[0,i-1])$.

Note that, by definition prefix average comparison is defined on inequality relation $\geq$

or $\leq$, and not for the other inequality or equality relations. Intuitively, prefix-average comparison states that $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$ if eventually the sum of prefixes of A are always greater than those of B. We use $\geq$ since the average of prefixes may be equal when the difference between the sum is small. It coincides with limit-average comparison when the limit-average exists for both sequences. Definition4.1 and Lemma 4.7-4.8 relate limit-average comparison and prefix-average comparison:

Corollary 4.3 *When limit-average of A and B exists, then*

- $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B) \implies \mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$.
- $\mathsf{LimAvg}(A) > \mathsf{LimAvg}(B) \implies \mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.

Proof 25 The first item falls directly from definitions.

For the second, let $\mathsf{LimAvgInf}(A)$ and $\mathsf{LimAvgSup}(B)$ be a and b respectively. For all $\epsilon > 0$ there exists an N_ϵ s.t for all $n > N_\epsilon$, $\frac{\mathsf{Sum}(A[1,n])}{n} > a - \epsilon$, and $\frac{\mathsf{Sum}(B[1,n])}{n} < b + \epsilon$. Let $a - b = k > 0$. Take $\epsilon = \frac{k}{4}$. Replicate the argument from Lemma 4.7 to show that there can exist only finitely many indexes i where $\mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$. Similarly, show there exists infinitely many prefixes where $\mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$ ■

Therefore, limit-average comparison and prefix-average comparison return the same result on sequences for which limit-average exists. In addition, prefix-average returns intuitive results when even when limit-average may not exist. For example, suppose limit-average of A and B do not exist, but $\mathsf{LimAvgInf}(A) > \mathsf{LimAvgSup}(B)$, then $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$. Therefore, prefix-average comparison relaxes limit-average comparison.

The rest of this section describes *prefix-average comparator for relation* $\geq$, denoted by $\mathcal{A}^{\geq}_{\mathsf{PA}}$, an automaton that accepts the pair (A, B) of sequences iff $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.

Lemma 4.9 (Pumping Lemma for ω-regular language [15]) *Let L be an ω-regular language. There exists $p \in \mathbb{N}$ such that, for each $w = u_1 w_1 u_2 w_2 \cdots \in L$ such that $|w_i| \geq p$ for all i, there are sequences of finite words $(x_i)_{i\in\mathbb{N}}$, $(y_i)_{i\in\mathbb{N}}$, $(z_i)_{i\in\mathbb{N}}$ s.t., for all i, $w_i = x_i y_i z_i$, $|x_i y_i| \leq p$ and $|y_i| > 0$ and for every sequence of pumping factors $(j_i)_{i\in\mathbb{N}} \in \mathbb{N}$, the pumped word $u_1 x_1 y_1^{j_1} z_1 u_2 x_2 y_2^{j_2} z_2 \cdots \in L$.*

Theorem 4.7 *The prefix-average comparator for $\geq$ is not ω-regular.*

Proof 26 We use Lemma 4.9 to prove that $\mathcal{A}^{\geq}_{\mathsf{PA}}$ is not ω-regular. Suppose $\mathcal{A}^{\geq}_{\mathsf{PA}}$ were ω-regular. For $p > 0 \in \mathbb{N}$, let $w = (A, B) = ((0,1)^p(1,0)^{2p})^\omega$. The segment $(0,1)^*$ can be pumped s.t the resulting word is no longer in $\mathcal{A}^{\geq}_{\mathsf{PA}}$.

Concretely, $A = (0^p 1^{2p})^\omega$, $B = (1^p 0^{2p})^\omega$, $\mathsf{LimAvg}(A) = \frac{2}{3}$, $\mathsf{LimAvg}(B) = \frac{1}{3}$. So, $w = (A, B) \in \mathcal{A}^{\geq}_{\mathsf{PA}}$. Select as factor w_i (from Lemma 4.9) the sequence $(0,1)^p$. Pump each y_i enough times so that the resulting word is $\hat{w} = (\hat{A}, \hat{B}) = ((0,1)^{m_i}(1,0)^{2p})^\omega$ where $m_i > 4p$. It is easy to show that $\hat{w} = (\hat{A}, \hat{B}) \notin \mathcal{A}^{\geq}_{\mathsf{PA}}$. ■

We discuss key ideas and sketch the construction of the prefix average comparator. The term *prefix-sum difference at i* indicates $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1])$, i.e. the difference between sum of i-length prefix of A and B.

Key ideas

For sequences A and B to satisfy $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$, $\exists^f i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$ and $\exists^\infty i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$. This occurs iff there

exists an index N s.t. for all indices $i > N$, $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1]) > 0$. While reading a word, the prefix-sum difference is maintained by states and the stack of ω-PDA: states maintain whether it is negative or positive, while number of tokens in the stack equals its absolute value. The automaton non-deterministically guesses the aforementioned index N, beyond which the automaton ensure that prefix-sum difference remains positive.

Construction sketch

The push-down comparator $\mathcal{A}^{\geq}_{\mathsf{PA}}$ consists of three states: (i) State s_P and (ii) State s_N that indicate that the prefix-sum difference is greater than zero and or not respectively, (iii) accepting state s_F. An execution of (A, B) begins in state s_N with an empty stack. On reading letter (a, b), the stack pops or pushes $|(a-b)|$ tokens from the stack depending on the current state of the execution. From state s_P, the stack pushes tokens if $(a-b) > 0$, and pops otherwise. The opposite occurs in state s_N. State transition between s_N and s_P occurs only if the stack action is to pop but the stack consists of $k < |a-b|$ tokens. In this case, stack is emptied, state transition is performed and $|a-b| - k$ tokens are pushed into the stack. For an execution of (A, B) to be an accepting run, the automaton non-deterministically transitions into state s_F. State s_F acts similar to state s_P except that execution is terminated if there aren't enough tokens to pop out of the stack. $\mathcal{A}^{\geq}_{\mathsf{PA}}$ accepts by accepting state.

To see why the construction is correct, it is sufficient to prove that at each index i, the number of tokens in the stack is equal to $|\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1])|$. Furthermore, in state s_N, $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1]) \leq 0$, and in state s_P and s_F, $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1]) > 0$. Next, the index at which the automaton transitions to the accepting state s_F coincides with index N. The

execution is accepted if it has an infinite execution in state s_F, which allows transitions only if $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1]) > 0$.

Construction

We provide a sketch of the construction of the Büchi push-down autoamaton $\mathcal{A}^{\geq}_{\mathsf{PA}}$, and then prove that it corresponds to the prefix average comparator.

Let μ be the bound on sequences. Then $\Sigma = \{0, 1, \ldots, n\}$ is the alphabet of sequences. Let $\mathcal{A}^{\geq}_{\mathsf{PA}} = (S, \Sigma \times \Sigma, \Gamma, \delta, s_0, Z_0)$ where:

- $S = \{s_N, s_P, s_F\}$ is the set of states of the automaton.
- $\Sigma \times \Sigma$ is the alphabet of the language.
- $\Gamma = \{Z_0, \alpha\}$ is the push down alphabet.
- $s_0 = s_N$ is the start state of the push down automata.
- Z_0 is the start symbol of the stack.
- s_F is the accepting state of the automaton. Automaton $\mathcal{A}^{\geq}_{\mathsf{PA}}$ accepts words by final state.
- Here we give a sketch of the behavior of the transition function δ.
  - When $\mathcal{A}^{\geq}_{\mathsf{PA}}$ is in configuration (s_P, τ) for $\tau \in \Gamma$, push a number of α-s into the stack.

    Next, pop b number of α-s. If after popping k α-s where $k < b$, the PDA's configuration becomes (s_P, Z_0), then first move to state (s_N, Z_0) and then resume with pushing $b - k$ α-s into the stack.

- When $\mathcal{A}^{\geq}_{\mathsf{PA}}$ is in configuration (s_N, τ) for $\tau \in \Gamma$, push b number of α-s into the stack

  Next, pop a number of α-s. If after popping k α-s where $k < a$, the PDA's configuration becomes (s_N, Z_0), then first move to state (s_P, Z_0) and then resume with pushing $a - k$ α-s into the stack.

- When $\mathcal{A}^{\geq}_{\mathsf{PA}}$ is in configuration (s_P, τ) for $\tau \neq Z_0$, first move to configuration (s_F, τ) and then push a number of α-s and pop b number of α-s. Note that there are no provisions for popping α if the stack hits Z_0 along this transition.

- When $\mathcal{A}^{\geq}_{\mathsf{PA}}$ is in configuration (s_F, τ) for $\tau \neq Z_0$, push a α-s then pop b α-s.

  Note that there are no provisions for popping α if the stack hits Z_0 along this transition.

Lemma 4.10 *Push down automaton $\mathcal{A}^{\geq}_{\mathsf{PA}}$ accepts a pair of sequences (A, B) iff* $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.

Proof 27 To prove this statement, it is sufficient to demonstrate that $\mathcal{A}^{\geq}_{\mathsf{PA}}$ accepts a pair of sequences (A, B) iff there are only finitely many indexes where $\mathsf{Sum}(B[1, i]) > \mathsf{Sum}(A[1, i])$. This is true by definition of $\mathsf{PrefixAvg}$ itself.

On $\mathcal{A}^{\geq}_{\mathsf{PA}}$ this corresponds to the condition that there being only finitely many times when the PDA is in state N during the run of (A, B). This is ensured by the push down automaton since the word can be accepted only in state F and there is no outgoing edge from F. Therefore, every word that is accepted by $\mathcal{A}^{\geq}_{\mathsf{PA}}$ satisfies the condition $\exists^f i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$.

Conversely, for every word (A, B) that satisfies $\exists^f i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$ there is a point, call it index k, such that for all indexes $m > k$, $\mathsf{Sum}(B[1, m]) \ngeq \mathsf{Sum}(A[1, m])$. If a run of (A, B) switches to F at this m, then it will be accepted by the push down automaton. Since $\mathcal{A}^{\geq}_{\mathsf{PA}}$ allows for non-deterministic move to (F, τ) from (P, τ), the run of (A, B) will always be able to move to F after index m. Hence, every (A, B) satisfying $\exists^f i, \mathsf{Sum}(B[0, i-1]) \geq \mathsf{Sum}(A[0, i-1])$ will be accepted by $\mathcal{A}^{\geq}_{\mathsf{PA}}$. ■

Theorem 4.8 *The prefix-average comparator for relation $\geq$ is an ω-CFL.*

While ω-CFL can be easily expressed, they do not possess closure properties, and problems on ω-CFL are easily undecidable. Hence, the application of ω-context-free comparator will require further investigation.

4.4 ω-Regular aggregate functions

So far, we have investigated aggregate functions individually in order to determine if their comparators are ω-regular. In this section, we investigate a broader class of aggregate functions, called ω-regular aggregate functions [42]. Intuitively, ω-regular functions are those aggregate functions for which arithmetic can be conducted on an automaton. Examples of ω-regular functions include discounted-sum with integer discount factors, limsup and liminf are examples of aggregate functions that are ω-regular.

The question we ask is whether a necessary and sufficient condition for an aggregate function to have an ω-regular compartor is that the function should be ω-regular? We prove one side of the argument. We show that if a function is ω-regular, then its comparator will be ω-regular. A proof/disproof of the other direction is open for

investigation.

The argument showing that every ω-regular function will have ω-regular comparators is given below:

Theorem 4.9 *Let $\mu > 0$ be the upper-bound on weight sequences, and $\beta \geq 2$ be the integer base. Let $f : \{0, 1, \ldots, \mu\}^\omega \to \mathbb{R}$ be an aggregate function. If aggregate function f is ω-regular under base β, then its comparator for all inequality and equality relations is also ω-regular.*

Proof 28 We show that if an aggregate function is ω-regular under base β, then its comparator for relation $>$ is ω-regular. By closure properties of ω-regular comparators, this implies that comparators of the aggregate function are ω-regular for all inequality and equality relations.

But first we prove that for a given integer base $\beta \geq 2$ there exists an automaton $\mathcal{A}_\beta$ such that for all $a, b \in \mathbb{R}$, $\mathcal{A}_\beta$ accepts $(\mathsf{rep}(a, \beta), \mathsf{rep}(b, \beta))$ iff $a > b$. Let $a, b \in \mathbb{R}$, and $\beta > 2$ be an integer base. Let $\mathsf{rep}(a, \beta) = \mathsf{sign}_a \cdot (\mathsf{Int}(a, \beta), \mathsf{Frac}(a, \beta))$ and $\mathsf{rep}(b, \beta) = \mathsf{sign}_b \cdot (\mathsf{Int}(b, \beta), \mathsf{Frac}(b, \beta))$. Then, the following statements can be proven using simple evaluation from definitions:

- When $\mathsf{sign}_a = +$ and $\mathsf{sign}_b = -$. Then $a > b$.
- When $\mathsf{sign}_a = \mathsf{sign}_b = +$
  - If $\mathsf{Int}(a, \beta) \neq \mathsf{Int}(b, \beta)$: Since $\mathsf{Int}(a, \beta)$ and $\mathsf{Int}(b, \beta)$ eventually only see digit 0 i.e. they are necessarily identical eventually, there exists an index i such that it is the last position where $\mathsf{Int}(a, \beta)$ and $\mathsf{Int}(b, \beta)$ differ. If $\mathsf{Int}(a, \beta)[i] > \mathsf{Int}(b, \beta)[i]$, then $a > b$. If $\mathsf{Int}(a, \beta)[i] < \mathsf{Int}(b, \beta)[i]$, then $a < b$.

  - If $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$ but $\mathsf{Frac}(a,\beta) \neq \mathsf{Frac}(b,\beta)$: Let i be the first index where $\mathsf{Frac}(a,\beta)$ and $\mathsf{Frac}(b,\beta)$ differ. If $\mathsf{Frac}(a,\beta)[i] > \mathsf{Frac}(b,\beta)[i]$ then $a > b$. If $\mathsf{Frac}(a,\beta)[i] < \mathsf{Frac}(b,\beta)[i]$ then $a < b$.
  - Finally, if $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$ and $\mathsf{Frac}(a,\beta) = \mathsf{Frac}(b,\beta)$: Then $a = b$.

- When $\mathsf{sign}_a = \mathsf{sign}_b = -$
  - If $\mathsf{Int}(a,\beta) \neq \mathsf{Int}(b,\beta)$: Since $\mathsf{Int}(a,\beta)$ and $\mathsf{Int}(b,\beta)$ eventually only see digit 0 i.e. they are necessarily identical eventually. Therefore, there exists an index i such that it is the last position where $\mathsf{Int}(a,\beta)$ and $\mathsf{Int}(b,\beta)$ differ. If $\mathsf{Int}(a,\beta)[i] > \mathsf{Int}(b,\beta)[i]$, then $a < b$. If $\mathsf{Int}(a,\beta)[i] < \mathsf{Int}(b,\beta)[i]$, then $a > b$.
  - If $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$ but $\mathsf{Frac}(a,\beta) \neq \mathsf{Frac}(b,\beta)$: Let i be the first index where $\mathsf{Frac}(a,\beta)$ and $\mathsf{Frac}(b,\beta)$ differ. If $\mathsf{Frac}(a,\beta)[i] > \mathsf{Frac}(b,\beta)[i]$ then $a < b$. If $\mathsf{Frac}(a,\beta)[i] < \mathsf{Frac}(b,\beta)[i]$ then $a > b$.
  - Finally, if $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$ and $\mathsf{Frac}(a,\beta) = \mathsf{Frac}(b,\beta)$: Then $a = b$.

- When $\mathsf{sign}_a = -$ and $\mathsf{sign}_b = +$. Then $a < b$.

Since the conditions given above are exhaustive and mutually exclusive, we conclude that for all $a, b \in \mathbb{R}$ and integer base $\beta \geq 2$, let $\mathsf{rep}(a,\beta) = \mathsf{sign}_a \cdot (\mathsf{Int}(a,\beta), \mathsf{Frac}(a,\beta))$ and $\mathsf{rep}(b,\beta) = \mathsf{sign}_b \cdot (\mathsf{Int}(b,\beta), \mathsf{Frac}(b,\beta))$. Then $a > b$ iff one of the following conditions occurs:

1. $\mathsf{sign}_a = +$ and $\mathsf{sign}_b = -$.
2. $\mathsf{sign}_a = \mathsf{sign}_b = +$, $\mathsf{Int}(a,\beta) \neq \mathsf{Int}(b,\beta)$, and $\mathsf{Int}(a,\beta)[i] > \mathsf{Int}(b,\beta)[i]$ when i is the last index where $\mathsf{Int}(a,\beta)$ and $\mathsf{Int}(b,\beta)$ differ.

3. $\mathsf{sign}_a = \mathsf{sign}_b = +$, $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$, $\mathsf{Frac}(a,\beta) \neq \mathsf{Frac}(b,\beta)$, and $\mathsf{Int}(a,\beta)[i] > \mathsf{Int}(b,\beta)[i]$ when i is the first index where $\mathsf{Frac}(a,\beta)$ and $\mathsf{Frac}(b,\beta)$ differ.

4. $\mathsf{sign}_a = \mathsf{sign}_b = +$, $\mathsf{Int}(a,\beta) \neq \mathsf{Int}(b,\beta)$, and $\mathsf{Int}(a,\beta)[i] < \mathsf{Int}(b,\beta)[i]$ when i is the last index where $\mathsf{Int}(a,\beta)$ and $\mathsf{Int}(b,\beta)$ differ.

5. $\mathsf{sign}_a = \mathsf{sign}_b = +$, $\mathsf{Int}(a,\beta) = \mathsf{Int}(b,\beta)$, $\mathsf{Frac}(a,\beta) \neq \mathsf{Frac}(b,\beta)$, and $\mathsf{Int}(a,\beta)[i] < \mathsf{Int}(b,\beta)[i]$ when i is the first index where $\mathsf{Frac}(a,\beta)$ and $\mathsf{Frac}(b,\beta)$ differ.

Note that each of these five condition can be easily expressed by a Büchi automaton over alphabet $\mathsf{AlphaRep}(\beta)$ for an integer $\beta \geq 2$. For an integer $\beta \geq 2$, the union of all these Büchi automata will result in a Büchi automaton $\mathcal{A}_\beta$ such that for all $a, b \in \mathbb{R}$ and $A = \mathsf{rep}(a,\beta)$ and $B = \mathsf{rep}(b,\beta)$, $a > b$ iff interleaved word $(A,B) \in \mathcal{L}(\mathcal{A}_\beta)$.

Now we come to the main part of the proof. Let $f : \Sigma^\omega \to \mathbb{R}$ be an ω-regular aggregate function with aggregate function automata $\mathcal{A}_f$. We will construct an ω-regular comparator for f with relation $>$. Note that (X,Y) is present in the comparator iff $(X,M),(Y,N) \in \mathcal{A}_f$ for $M, N \in \mathsf{AlphaRep}(\beta)^\omega$ and $(M,N) \in \mathcal{A}_\beta$, for $\mathcal{A}_\beta$ as described above. Since $\mathcal{A}_f$ and $\mathcal{A}_\beta$ are both Büchi automata, the comparator for function f with relation $>$ is also a Büchi auotmaton. Therefore, the comparator for aggregate function f with relation $>$ is ω-regular. ■

The converse direction is still open For all aggregate functions considered in this paper for which the comparator is ω-regular, the function has also been ω-regular. But that is not a full proof of the converse direction. For now, due to the lack of any counterexample, we present the converse direction as a conjecture:

Conjecture 4.1 *Let $\mu > 0$ be the upper-bound on weight sequences, and $\beta \geq 2$ be the*

integer base. Let $f : \{0, 1, \ldots, \mu\}^\omega \to \mathbb{R}$ *be an aggregate function. If the comparator for an aggregate function* f *is* ω*-regular for all inequality and equality relations, then its aggregate function is also* ω*-regular under base* β*.*

4.5 Chapter summary

This chapter studied comparator automata for well known aggregate functions, namely discounted-sum and limit average. Among these, only discounted-sum with integer discount factors can be represented by ω-regular comparators. In later chapters, we will observe that once can design ω-regular comparators for approximations to discounted-sum with non-integer discount factor (Chapter 7). To obtain a a broader classification of aggregate functions that permit ω-regular comparators, we show that ω-regular aggregate functions exhibit ω-regular comparators. However, we do not whether ω-regular functions is the sufficient condition for the existence of ω-regular comparators. We conjecture that may be the case, but leave that as an open question.

Part II

Quantitative inclusion with discounted-sum

Having laid out the theoretical framework of comparator automata in Part I, Part II and Part III demonstrate the efficacy of the integrated approach proposed by comparators.

This Chapter 3 studies Discounted-sum inclusion or DS inclusion in detail. Recall, in Theorem 4.5 we use comparator automata to establish that DS inclusion is PSPACE-complete when the discount factor is an integer; The decidability of DS inclusion is still unknown when the discount factor is not an integer. This part will delve into solving DS inclusion in practice.

Chapter 5 conducts the first comparative study of the empirical performance of existing algorithms for DS inclusion with integer discount factors. These are a separation-of-techniques algorithm from prior work, and our ω-regular comparator algorithm for DS inclusion. Our analysis shows how the two approaches complement each other. This is a nuanced picture that is much richer than the one obtained from the complexity-theoretic study alone.

Chapter 6 picks up from where Chapter 5 ends. We prove that DS comparison languages are either safety or co-safety langauges. We show that when this property is utilized to solve DS inclusion using comparators, then comparator-based approach outperforms the separation-of-techniques algorithm on all accounts.

Chapter 7 shifts focus to solving DS inclusion with non-integer discount factors. Currently even its decidability is unknown. Therefore, to solve DS inclusion in practice, we design an *anytime algorithm*, which will either terminate with a crisp True or False answer after a finite amount of time, or continuously generate a tighter approximation. This algorithm makes use of comparator automata for approximations of DS with non-integer discount factor, which are shown to

exhibit regularity.

Terminology and notation

We refer to a weighted ω-automata with the discounted-sum aggregate function by *discounted-sum automata*. In detail, a *discounted-sum automaton* with discount factor $d > 1$, *DS automaton* in short, is a tuple $\mathcal{A} = (\mathcal{M}, \gamma)$, where $\mathcal{M} = (S, \Sigma, \delta, Init, S)$ is a Büchi automaton, and $\gamma : \delta \to \mathbb{N}$ is the *weight function* that assigns a weight to each transition of automaton $\mathcal{M}$. *Words* and *runs* in weighted ω-automata are defined as they are in Büchi automata. Note that all states are accepting states in this definition. The *weight sequence* of run $\rho = s_0 s_1 \dots$ of word $w = w_0 w_1 \dots$ is given by $wt_\rho = n_0 n_1 n_2 \dots$ where $n_i = \gamma(s_i, w_i, s_{i+1})$ for all i. The *weight of a run* ρ is given by $DS(wt_\rho, d)$. For simplicity, we denote this by $DS(\rho, d)$. The *weight of a word* in DS automata is defined as $wt_{\mathcal{A}}(w) = sup\{DS(\rho, d) | \rho \text{ is a run of } w \text{ in } \mathcal{A}\}$. By convention, if a word $w \notin \mathcal{L}(\mathcal{A})$, then $wt_{\mathcal{A}}(w) = 0$ [39]. A DS automata is said to be *complete* if from every state there is at least one transition on every alphabet. Formally, for all $p \in S$ and for all $a \in \Sigma$, there exists $q \in S$ s.t $(p, a, q) \in \delta$. We abuse notation, and use $w \in \mathcal{A}$ to mean $w \in \mathcal{L}(\mathcal{A})$ for Büchi automaton or DS-automaton $\mathcal{A}$. Given DS automata P and Q and discount-factor $d > 1$, the *discounted-sum inclusion problem*, denoted by $P \subseteq_d Q$, determines whether for all words $w \in \Sigma^\omega$, $wt_P(w) \leq wt_Q(w)$.

Chapter 5

Analysis of DS inclusion with integer discount factor

Discounted-sum inclusion, or DS inclusion, is when quantitative inclusion is performed with the discounted-sum aggregation function. Prior work has demonstrated the applicability of DS inclusion in computation of rational solutions in multi-agent systems with rational agents [23]. Yet, the focus on DS inclusion has mostly been from a complexity-theoretic perspective, and not on algorithmic performance. Even in this thesis so far, comparator automata have been used to establish that DS inclusion with integer discount factor is PSPACE-complete. But whether these comparator-based algorithms are scalable and efficient has not been evaluated yet.

To this end, this chapter undertakes a thorough theoretical and empirical analysis of two contrasting approaches for DS inclusion with integer discount factors: our comparator-based integrated algorithm, and the prior known separation-of-techniques algorithm. We present the first implementations of these algorithms, and perform extensive experimentation to compare between the two approaches. Our analysis shows how the two approaches complement each other. This is a nuanced picture that is much richer than the one obtained from the complexity-theoretic study alone.

5.1 Saga of theoretical vs empirical analysis

The hardness of quantitative inclusion for nondeterministic DS automata, or DS inclusion, is evident from PSPACE-hardness of language-inclusion (LI) problem for

nondeterministic Büchi automata [87]. Decision procedures for DS inclusion were first investigated in [39], and subsequently through target discounted-sum [32], DS-determinization [31]. The comparator-based argument [25], presented in Chapter 4, finally established its PSPACE-completeness. However, these theoretical advances in DS inclusion have not been accompanied with the development of efficient and scalable tools and algorithms. This is the focus of this chapter; our goal is to develop practical algorithms and tools for DS inclusion.

Theoretical advances have lead to two algorithmic approaches for DS inclusion. The first approach, referred to as DetLP, combines automata-theoretic reasoning with linear-programming (LP). This method first determinizes the DS automata [31], and reduces the problem of DS inclusion for deterministic DS automata to LP [17, 18]. Since determinization of DS automata causes an exponential blow-up, DetLP yields an exponential time algorithm. An essential feature of this approach is the separation of automata-theoretic reasoning– determinization–and numerical reasoning, performed by an LP-solver. Because of this separation, it does not seem easy to apply on-the-fly techniques to this approach and perform it using polynomial space, so this approach uses exponential time and space.

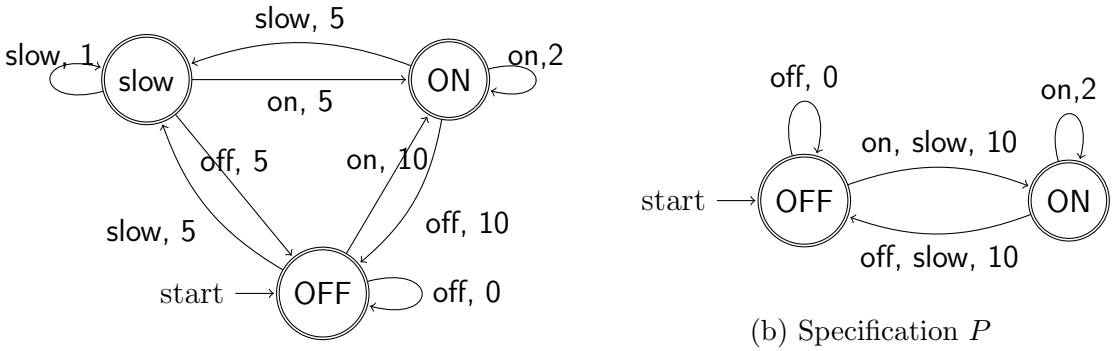
In contrast, the second algorithm for DS inclusion, referred to as BCV (after name of authors) is purely automata-theoretic [25], was presented in Chapter 4-Section 4.2.1. The component of numerical reasoning between costs of executions is handled by a special Büchi automaton, called the *comparator*, that enables an on-line comparison of the discounted-sum of a pair of weight-sequences. Aided by the comparator, BCV reduces DS inclusion to language-equivalence between Büchi automata. Since language-equivalence is in PSPACE, BCV is a polynomial-space algorithm.

While the complexity-theoretic argument may seem to suggest a clear advantage

for the pure automata-theoretic approach of BCV, the perspective from an implementation point of view is more nuanced. BCV relies on LI-solvers as its key algorithmic component. The polynomial-space approach for LI relies on Savitch's Theorem, which proves the equivalence between deterministic and non-deterministic space complexity [81]. This theorem, however, does not yield a practical algorithm. Existing efficient LI-solvers [4, 5] are based on Ramsey-based inclusion testing [7] or rank-based approaches [64]. These tools actually use exponential time and space. In fact, the exponential blow-up of Ramsey-based approach seems to be worse than that of DS-determinization. Thus, the theoretical advantage BCV seems to evaporate upon close examination. Thus, it is far from clear which algorithmic approach is superior. To resolve this issue, we provide in this paper the first implementations for both algorithms and perform exhaustive empirical analysis to compare their performance.

Our first tool, also called DetLP, implements its namesake algorithm as it is. We rely on existing LP-solver GLPSOL to perform numerical reasoning. Our second tool, called QuIP, starts from BCV, but improves on it. The key improvement arises from the construction of an improved comparator with fewer states. We revisit the reduction to language inclusion in [25] accordingly. The new reduction reduces the transition-density of the inputs to the LI-solver (Transition density is the ratio of transitions to states), improving the overall performance of QuIP since LI-solvers are known to scale better at lower transition-density inputs [71]

Our empirical analysis reveals that theoretical complexity does not provide a full picture. Despite its poorer complexity, QuIP scales significantly better than DetLP, although DetLP solves more benchmarks. Based on these observations, we propose a method for DS inclusion that leverages the complementary strengths of these tools to offer a scalable tool for DS inclusion. Our evaluation also exposes the limitations of



(a) System S

(b) Specification P

Figure 5.1 : DS inclusion: Motivating example

both approaches, and opens up avenues for improvement in tools for DS inclusion.

Motivating example

As an example of such a problem formulation, consider the system and specification in Figure 5.1a and Figure 5.1b, respectively [39]. Here, the specification P depicts the worst-case energy-consumption model for a motor, and the system S is a candidate implementation of the motor. Transitions in S and P are labeled by transition-action and transition-cost. The cost of an execution (a sequence of actions) is given by an *aggregate* of the costs of transitions along its run (a sequence of automaton states). In non-deterministic automata, where each execution may have multiple runs, cost of the execution is the cost of the run with maximum cost. A critical question here is to check whether implementation S is more energy-efficient than specification P. This problem can be framed as a problem of quantitative inclusion between S and P.

5.2 Theoretical analysis of existing algorithms

A purely complexity-theoretic analysis will indicate that DetLP will fare poorer than BCV in practice since the former is exponential in time and space whereas the later is only polynomial in space. However, an empirical evaluation of these algorithms reveals that opposite: DetLP outperforms BCV .

This section undertakes a closer theoretical examination of both algorithms in order to shed light on the seemingly anomalous behavior. Not only does our analysis explain the behavior but also uncovers avenues for improvements to BCV .

5.2.1 DetLP: DS determinization and Linear programming

DetLP follows a separation-of-techniques approach wherein the first step consists of determinization of the DS automata and the second step reduces DS inclusion to linear programming. As one may notice, the first step is automata-based while the second is based on numerical methods.

Böker and Henzinger studied complexity and decision-procedures for determinization of DS automata in detail [31]. They proved that a DS automata can be determinized if it is complete, all its states are accepting states and the discount-factor is an integer. Under all other circumstances, DS determinization may not be guaranteed. DS determinization extends subset-construction for automata over finite words. Every state of the determinized DS automata is represented by an $|S|$-tuple of numbers, where $S = \{q_1, \ldots q_{|S|}\}$ denotes the set of states of the original DS-automaton. The value stored in the i-th place in the $|S|$-tuple represents the "gap" or extra-cost of reaching state q_i over a finite-word w compared to its best value so far. The crux of the argument lies in proving that when the DS automata is complete and the discount-factor is an integer, the "gap" can take only finitely-many values, yielding finiteness

of the determinized DS automata, albeit exponentially larger than the original.

Theorem 5.1 [31] [DS determinization analysis] *Let A be a complete DS automata with maximum weight μ over transitions and s number of states. DS determinization of A generates a DS-automaton with at most μ^s states.*

Chatterjee et al. reduced $P \subseteq_d Q$ between non-deterministic DS automata P and deterministic DS automata Q to linear-programming [17,18,39]. First, the product DS automata $P \times Q$ is constructed so that $(s_P, s_Q) \xrightarrow{a} (t_P, t_Q)$ is a transition with weight $w_P - w_Q$ if transition $s_M \xrightarrow{a} t_M$ with weight w_M is present in M, for $M \in \{P, Q\}$. $P \subseteq_q Q$ is False iff the weight of any word in $P \times Q$ is greater than 0. Since Q is deterministic, it is sufficient to check if the maximum weight of all infinite paths from the initial state in $P \times Q$ is greater than 0. For discounted-sum, the maximum weight of paths from a given state can be determined by a linear-program: Each variable (one for each state) corresponds to the weight of paths originating in this state, and transitions decide the constraints which relate the values of variables (or states) on them. The objective is to maximize weight of variable corresponding to the initial state.

Therefore, the DetLP method for $P \subseteq_d Q$ is as follows: Determinize Q to Q_D via DS determinization method from [31], and reduce $P \subseteq_d Q_D$ to linear programming following [39]. Note that since determinization is possible only if the DS automaton is complete, DetLP can be applied only if Q is complete.

Lemma 5.1 *Let P and Q be non-deterministic DS automata with s_P and s_Q number of states respectively, τ_P states in P. Let the alphabet be Σ and maximum weight on transitions be μ. Then $P \subseteq_d Q$ is reduced to linear programming with $\mathcal{O}(s_P \cdot \mu^{s_Q})$ variables and $\mathcal{O}(\tau_P \cdot \mu^{s_Q} \cdot |\Sigma|)$ constraints.*

Therefore, the final complexity of DetLP is as follows:

Theorem 5.2 [17, 39] [Complexity of DetLP] *Let P and Q be DS automata with s_P and s_Q number of states respectively, τ_P states in P. Let the alphabet be Σ and maximum weight on transitions be μ. Complexity of DetLP is $\mathcal{O}(s_P^2 \cdot \tau_P \cdot \mu^{s_Q} \cdot |\Sigma|)$.*

Proof 29 Anderson and Conitzer [17] proved that this system of linear equations can be solved in $\mathcal{O}(m \cdot n^2)$ for m constraints and n variables. ■

5.2.2 BCV: Comparator-based approach

BCV is based on an integrated approach for DS inclusion wherein the entire algorithm uses automata-based reasoning only. Strictly speaking, BCV is based on a generic algorithm for inclusion under a general class of aggregate functions which permit ω-regular comparators, presented in Chapter 4-Section 4.2.1. BCV (Algorithm 2) refers to its adaptation to DS. It is described in complete detail, for sake of clarity.

Recall, a run $\rho \in P$ of word $w \in \mathcal{L}(P)$ is a said to be *dominated w.r.t* Q if there exists a run $\sigma \in Q$ over the same word w s.t. $DS(\rho, d) < DS(\sigma, d)$. The key idea behind BCV is that $P \subseteq_d Q$ holds iff every run of P is a dominated run w.r.t Q. As a result, BCV constructs an intermediate Büchi automaton *Dom* that consists of all dominated runs of P w.r.t Q. It then checks whether *Dom* consists of all runs of P, by determining language-equivalence between *Dom* and an automaton $\hat{P}$ that consists of all runs of P. The comparator $\mathcal{A}_{\leq}^{\mu,d}$ is utilized in the construction of *Dom* to compare weight of runs in P and Q.

Procedure AugmentWtAndLabel separates between runs of the same word in DS automata by assigning a unique transition-identity to each transition. It also appends the transition weight, to enable weight comparison afterwards. Specifically, it

Algorithm 2 BCV(P, Q, d), Is $P \subseteq_d Q$?

1: **Input:** Weighted automata P, Q, and discount-factor d

2: **Output:** True if $P \subseteq_d Q$, False otherwise

3: $\hat{P} \leftarrow$ AugmentWtAndLabel(P)

4: $\hat{Q} \leftarrow$ AugmentWtAndLabel(Q)

5: $\hat{P} \times \hat{Q} \leftarrow$ MakeProductSameAlpha$(\hat{P}, \hat{Q})$

6: $\mu \leftarrow$ MaxWeight(P, Q)

7: $\mathcal{A}_{\leq}^{\mu,d} \leftarrow$ MakeComparator(μ, d)

8: $DomWithWitness \leftarrow$ Intersect$(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$

9: $Dom \leftarrow$ FirstProject$(DomWithWitness)$

10: **return** $\hat{P} \equiv Dom$

transforms DS-automaton $\mathcal{A}$ into Büchi automaton $\hat{\mathcal{A}}$, with all states as accepting, by converting transition $\tau = s \xrightarrow{a} t$ with weight wt and unique transition-identity l to transition $\hat{\tau} = s \xrightarrow{(a,wt,l)} t$ in $\hat{\mathcal{A}}$. Procedure MakeProductSameAlpha$(\hat{P}, \hat{Q})$ takes the product of $\hat{P}$ and $\hat{Q}$ over the same word i.e., transitions $s_{\mathcal{A}} \xrightarrow{(a,n_{\mathcal{A}},l_{\mathcal{A}})} t_{\mathcal{A}}$ in $\mathcal{A}$, for $\mathcal{A} \in \{\hat{P}, \hat{Q}\}$, generates transition $(s_P, s_Q) \xrightarrow{(a,n_P,l_P,n_Q,l_Q)} (t_P, t_Q)$ in $\hat{P} \times \hat{Q}$. The comparator $\mathcal{A}_{\leq}^{\mu,d}$ is constructed with upper-bound μ that equals the maximum weight of transitions in P and Q, and discount-factor d. Intersect matches the alphabet of $\hat{P} \times \hat{Q}$ with $\mathcal{A}_{\leq}^{\mu,d}$, and intersects them. The resulting automaton $DomWithWitness$ accepts word $(w, wt_P, id_P, wt_Q, id_Q)$ iff $DS(wt_P, d) \leq DS(wt_Q, d)$. The projection of $DomWithWitness$ on the first three components of $\hat{P}$ returns Dom which contains the word (w, wt_P, id_P) iff it is a dominated run in P. Finally, language-equivalence between Dom and $\hat{P}$ returns the answer.

Analysis of BCV

The proof for PSPACE-complexity of BCV relies on LI to be PSPACE. In practice, though, implementations of LI apply Ramsey-based inclusion testing [7], rank-based methods [64] etc. All of these algorithms are exponential in time and space in the worst case. Any implementation of BCV will have to rely on an LI-solver. Therefore, in practice BCV is also exponential in time and space. In fact, we show that its worst-case complexity (in practice) is poorer than DetLP.

Another reason that prevents BCV from practical implementations is that it does not optimize the size of intermediate automata. Specifically, we show that the size and transition-density of *Dom*, which is one of the inputs to LI-solver, is very high (Transition density is the ratio of transitions to states). Both of these parameters are known to be deterrents to the performance of existing LI-solvers [6], subsequently to BCV as well:

Lemma 5.2 *Let s_P, s_Q, s_d and τ_P, τ_Q, τ_d denote the number of states and transitions in P, Q, and $\mathcal{A}_{\leq}^{\mu,d}$, respectively. Number of states and transitions in Dom are $\mathcal{O}(s_P s_Q s_d)$ and $\mathcal{O}(\tau_P^2 \tau_Q^2 \tau_d |\Sigma|)$, respectively.*

Proof 30 It is easy to see that the number of states and transitions of $\hat{P}$ $\hat{Q}$ are the same as those of P and Q, respectively. Therefore, the number of states and transitions in $\hat{P} \times \hat{Q}$ are $\mathcal{O}(s_P s_Q)$ and $\mathcal{O}(\tau_P \tau_Q)$, respectively. The alphabet of $\hat{P} \times \hat{Q}$ is of the form $(a, wt_1, id_1, wt_2, id_2)$ for $a \in \Sigma$, wt_1, wt_2 are non-negative weights bounded by μ and id_i are unique transition-ids in P and Q respectively. The alphabet of comparator $\mathcal{A}_{\leq}^{\mu,d}$ is of the form (wt_1, wt_2). To perform intersection of these two, the alphabet of comparator needs to be matched to that of the product, causing a blow-up in number of transitions in the comparator by a factor of $|\Sigma| \cdot \tau_P \cdot \tau_Q$. Therefore, the

number of states and transitions in *DomWithWitness* and *Dom* is given by $\mathcal{O}(s_P s_Q s_d)$ and $\mathcal{O}(\tau_P^2 \tau_Q^2 \tau_d |\Sigma|)$. ■

The comparator is a non-deterministic Büchi automata with $\mathcal{O}(\mu^2)$ states over an alphabet of size μ^2 [25]. Since transition-density $\delta = |S| \cdot |\Sigma|$ for non-deterministic Büchi automata, the transition-density of the comparator is $\mathcal{O}(\mu^4)$. Therefore,

Corollary 5.1 *Let s_P, s_Q, s_d denote the number of states in P, Q, $\mathcal{A}_{\leq}^{\mu,d}$, respectively, and δ_P, δ_Q and δ_d be their transition-densities. Number of states and transition-density of Dom are $\mathcal{O}(s_P s_Q \mu^2)$ and $\mathcal{O}(\delta_P \delta_Q \tau_P \tau_Q \cdot \mu^4 \cdot |\Sigma|)$, respectively.*

The corollary illustrates that the transition-density of *Dom* is very high even for small inputs. The blow-up in number of transitions of *DomWithWitness* (hence *Dom*) occurs during alphabet-matching for Büchi automata intersection (Algorithm 2, Line 3). However, the blow-up can be avoided by performing intersection over a substring of the alphabet of $\hat{P} \times \hat{Q}$. Specifically, if $s_1 \xrightarrow{(a, n_P, id_P, n_Q, id_Q)} s_2$ and $t_1 \xrightarrow{(wt_1, wt_2)} t_2$ are transitions in $\hat{P} \times \hat{Q}$ and comparator $\mathcal{A}_{\leq}^{\mu,d}$ respectively, then $(s_1, t_1, i) \xrightarrow{(a, n_P, id_P, n_Q, id_Q)} (s_2, t_2, j)$ is a transition in the intersection iff $n_P = wt_1$ and $n_Q = wt_2$, where $j = (i + 1) \mod 2$ if either s_1 or t_1 is an accepting state, and $j = i$ otherwise. We call intersection over substring of alphabet Intersect. The following is easy to prove:

Lemma 5.3 *Let $\mathcal{A}_1 =$ Intersect$(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$, and $\mathcal{A}_2 =$ Intersect$(\hat{P} \times \hat{Q}, \mathcal{A}_{\leq}^{\mu,d})$.* Intersect *extends alphabet of $\mathcal{A}_{\leq}^{\mu,d}$ to match the alphabet of $\hat{P} \times \hat{Q}$ and* Intersect *selects a substring of the alphabet of $\hat{P} \times \hat{Q}$ as defined above. Then, $\mathcal{L}(\mathcal{A}_1) \equiv \mathcal{L}(\mathcal{A}_2)$.*

Intersect prevents the blow-up by $|\Sigma| \cdot \tau_P \cdot \tau_Q$, resulting in only $\mathcal{O}(\tau_P \tau_Q \tau_d)$ transitions in *Dom* Therefore,

Lemma 5.4 [Trans. Den. in BCV] *Let δ_P, δ_Q denote transition-densities of P and Q, resp., and μ be the upper bound for comparator $\mathcal{A}^{\mu,d}_{\leq}$. Number of states and transition-density of Dom are $\mathcal{O}(s_P s_Q \mu^2)$ and $\mathcal{O}(\delta_P \delta_Q \cdot \mu^4)$, respectively.*

Language equivalence is performed by tools for language inclusion. The most effective tool for language-inclusion RABIT [5] is based on Ramsay-based inclusion testing [7]. The worst-case complexity for $A \subseteq B$ via Ramsay-based inclusion testing is known to be $2^{\mathcal{O}(n^2)}$, when B has n states. Therefore,

Theorem 5.3 [Practical complexity of BCV] *Let P and Q be DS automata with s_P, s_Q number of states respectively, and maximum weight on transitions be μ. Worst-case complexity for BCV for integer discount-factor $d > 1$ when language-equivalence is performed via Ramsay-based inclusion testing is $2^{\mathcal{O}(s_P^2 \cdot s_Q^2 \cdot \mu^4)}$.*

Recall that language-inclusion queries are $\hat{P} \subseteq Dom$ and $Dom \subseteq \hat{P}$. Since Dom has many more states than $\hat{P}$, the complexity of $\hat{P} \subseteq Dom$ dominates.

Theorem 5.2 and Theorem 5.3 demonstrate that the complexity of BCV (in practice) is worse than DetLP. This explains the inferior performance of BCV in practice.

5.3 QuIP: Optimized BCV-based solver for DS inclusion

The ealrier investigate explains why BCV does not lend itself to a practical implementation for DS inclusion (§ 5.2.2), and also identifies its drawbacks. This section proposes an improved comparator-based algorithm QuIP for DS inclusion, as is described in § 5.3.2. QuIP improves upon BCV by means of a new optimized comparator that we describe in §5.3.1.

5.3.1 An optimized DS comparator

The $2^{\mathcal{O}(s^2)}$ dependence of BCV on the number of states s of the DS comparator motivates us to construct a more compact comparator. Currently a DS comparator consists of $\mathcal{O}(\mu^2)$ number of states for upper bound μ [25]. In this section, we re-define DS comparison languages and DS comparators so that they consist of only $\mathcal{O}(\mu)$-many states and have a transition density of $\mathcal{O}(\mu^2)$.

Definition 5.1 (DS comparison language) For an integer upper bound $\mu > 0$, discount factor $d > 1$, and equality or inequality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, the *DS comparison language with upper bound μ, relation* R*, and discount factor d* is a language of infinite words over the alphabet $\Sigma = \{-\mu, \ldots, \mu\}$ that accepts $A \in \Sigma^\omega$ iff $DS(A, d)$ R 0 holds.

Definition 5.2 (DS comparator automata) For an integer upper bound $\mu > 0$, discount factor $d > 1$, and equality or inequality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, the *DS comparator automata with upper bound μ, relation* R*, and discount factor d* is an automaton that accepts the DS comparison language with upper bound μ, relation R, and discount factor d.

Semantically, Definition 5.1 and Definition 5.2 with upper bound μ, discount-factor d and inequality relation R is the language and automaton, respectively, of all integer sequences bounded by μ for which their discounted-sum is related to 0 by the relation R. They differ from the definitions presented in Chapter 4-Section 4.2 since those definitions relate a pair of integer sequences with each other, while the current definitions related one sequence with the constant value 0. However, these definitions are equivalent since $DS(A, d) \leq DS(B, d) \equiv DS(A - B, d) \leq 0$, where the sequence $(A - B)$ refers to the sequence generated by taking a point-wise difference of elements

in A and B. Now onwards, we will always use the new definitions for DS comparison languages and DS comparators.

Note that this equivalence does not hold for all aggregation functions such as limsup and liminf. Hence, this modified definition for DS comparison languages and DS comparators specifically apply to the discounted-sum aggregation function. Another repercussion of thi equivalence is that the results on ω-regularity of DS comparison languages and comparators on the older definitions from Chapter 4-Section 4.1- 4.2 apply to these new definitions as well. Therefore, we obtain that DS comparison languages are ω-regular iff the discount factor is an integer. In fact, the automaton for DS comparator with upper bound μ, integer discount factor $d > 1$ and relation R, denoted by $\mathcal{B}_{\mathsf{R}}^{\mu,d}$ under Definition 5.2 can be derived from the DS comparator with the same parameters from the old definition, denoted $\mathcal{A}_{\mathsf{R}}^{\mu,d}$, by transforming the alphabet from (a, b) to $(a-b)$ along every transition. The first benefit of the modified alphabet is that its size is reduced from μ^2 to $2 \cdot \mu - 1$. In addition, it coalesces all transitions between any two states over alphabet $(a, a + v)$, for all a, into one single transition over v, thereby also reducing transitions. However, this direct transformation results in a comparator with $\mathcal{O}(\mu^2)$ states. This section presents a new construction of the comparator with $\mathcal{O}(\mu)$ states only.

In principle, the current construction adapts the construction from Chapter 4-Section 4.2 to the modified alphabett. The key idea behind the construction of the ew DS comparator, similar to the earlier construction, is that the discounted-sum of sequence V can be treated as a number in base d i.e. $DS(V, d) = \Sigma_{i=0}^{\infty} \frac{V[i]}{d^i} = (V[0].V[1]V[2]\dots)_d$. So, there exists a non-negative value C in base d s.t. $V + C = 0$ for arithmetic operations in base d. This value C can be represented by a non-negative sequence C s.t. $DS(C, d) + DS(V, d) = 0$. Arithmetic in base d over sequences C and

V result in a sequence of carry-on X such that:

Lemma 5.5 *Let V, C, X be the number sequences, $d > 1$ be a positive integer such that following equations holds true:*

1. *When $i = 0$, $V[0] + C[0] + X[0] = 0$*
2. *When $i \geq 1$, $V[i] + C[i] + X[i] = d \cdot X[i-1]$*

Then $DS(V, d) + DS(C, d) = 0$.

Proof 31 This follows proof by expansion of all terms. Specifically, expand out $DS(V, d) + DS(C, d)$ to $\Sigma_{i=0}^{\infty} \frac{(V[i]+C[i])}{d^i}$ and replace each $(V[i]+C[i])$ with substitutions given by the equations. Most terms will cancel each other, and by re-arrangement we will get $DS(V, d) + DS(C, d) = 0$. ■

In the construction of the comparator in Chapter 4-Section 4.2, it has been proven that when A and B are bounded non-negative integer sequences s.t. $DS(A, d) \leq DS(B, d)$, the corresponding sequences C and X are also bounded integer-sequences [25]. The same argument transcends here: When V is a bounded integer sequence s.t. $DS(V, d) \leq 0$, there exists a corresponding pair of bounded integer sequence C and X. In fact, the bounds used for the comparator carry over to this case as well. Sequence C is non-negative and is bounded by $\mu_C = \mu \cdot \frac{d}{d-1}$ since $-\mu_C$ is the minimum value of discounted-sum of V, and integer-sequence X is bounded by $\mu_X = 1 + \frac{\mu}{d-1}$. On combining Lemma 5.5 with the bounds on X and C we get:

Lemma 5.6 *Let V and be an integer-sequence bounded by μ s.t. $DS(V, d) \leq 0$, and X be an integer sequence bounded by $(1 + \frac{\mu}{d-1})$, then there exists an X s.t.*

1. *When $i = 0$, $0 \leq -(X[0] + V[0]) \leq \mu \cdot \frac{d}{d-1}$*

2. *When* $i \geq 1$, $0 \leq (d \cdot X[i-1] - V[i] - X[i]) \leq \mu \cdot \frac{d}{d-1}$

Proof 32 Equations 1-2 from Lemma 5.6 have been obtained by expressing $C[i]$ in terms of $X[i]$, $X[i-1]$, $V[i]$ and d, and imposing the non-negative bound of $\mu_C = \mu \cdot \frac{d}{d-1}$ on the resulting expression. Therefore, Lemma 5.6 implicitly captures the conditions on C by expressing it only in terms of V, X and d for $DS(V, d) \leq 0$ to hold. ■

In construction of the new DS comparator, the values of $V[i]$ is part of the alphabet, upper bound μ and discount-factor d are the input parameters. The only unknowns are the value of $X[i]$. However, we know that it can take only finitely many values i.e. integer values $|X[i]| \leq \mu_X$. So, we store all possible values of $X[i]$ in the states. Hence, the state-space S comprises of $\{(x)||x| \leq \mu_X\}$ and a start state s. Transitions between these states are possible iff the corresponding x-values and alphabet v satisfy the conditions of Equations 1-2 from Lemma 5.6. There is a transition from start state s to state (x) on alphabet v if $0 \leq -(x+v) \leq \mu \cdot \frac{d}{d-1}$, and from state (x) to state (x') on alphabet v if $0 \leq (d \cdot x - v - x') \leq \mu \cdot \frac{d}{d-1}$. All (x)-states are accepting. This completes the construction for DS comparator $\mathcal{B}_{\leq}^{\mu,d}$. Clearly $\mathcal{B}_{\leq}^{\mu,d}$ has only $\mathcal{O}(\mu)$ states. The formal construction is given below:

Construction

Let $\mu_C = \mu \cdot \frac{d}{d-1} \leq 2 \cdot \mu$ and $\mu_X = 1 + \frac{\mu}{d-1}$. $\mathcal{B}_{<}^{\mu,d} = (S, \Sigma, \delta_d, \mathit{Init}, \mathcal{F})$

- $S = \mathit{Init} \cup \mathcal{F} \cup S_{\perp}$ where

  $\mathit{Init} = \{s\}$, $\mathcal{F} = \{x||x| \leq \mu_X\}$, and

  $S_{\perp} = \{(x, \perp)||x| \leq \mu_X\}$ where $\perp$ is a special character, and $x \in \mathbb{Z}$.

- $\Sigma = \{v : |v| \leq \mu\}$ where v is an integer.

- $\delta_d \subset S \times \Sigma \times S$ is defined as follows:

  1. Transitions from start state s:

     i (s, v, x) for all $x \in \mathcal{F}$ s.t. $0 < -(x + v) \leq \mu_C$

     ii $(s, v, (x, \perp))$ for all $(x, \perp) \in S_\perp$ s.t. $x + v = 0$

  2. Transitions within $S_\perp$: $((x, \perp), v, (x', \perp))$ for all $(x, \perp)$, $(x', \perp) \in S_\perp$, if $d \cdot x = v + x'$

  3. Transitions within $\mathcal{F}$: (x, v, x') for all $x, x' \in \mathcal{F}$ if $0 \leq d \cdot x - v - x' < d$

  4. Transition between $S_\perp$ and $\mathcal{F}$: $((x, \perp), v, x')$ for $(x, \perp) \in S_\perp$, $x' \in \mathcal{F}$ if $0 < d \cdot x - v - x' < d$

Theorem 5.4 *The Büchi automaton $\mathcal{B}^{\mu,d}_{<}$ constructed above is DS comparator automata with with upper bound μ, integer discount factor $d > 1$ and relation $<$. DS comparator $\mathcal{B}^{\mu,d}_{<}$ consists of $\mathcal{O}(\mu)$-states, and a transition-density of $\mathcal{O}(\mu^2)$.*

Since Büchi automata are closed under set-theoretic operations, a counterpart of closure of ω-regular comparison languages under all relations (Theorem 3.1) applies to the new definition as well. Furthermore, simple modifications to the automaton constructed above will result in the automaton for all other relations. As a result, The DS comparator automata wth upper bound $\mu > 0$, integer discount factor $d > 1$ and relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$ will have $\mathcal{O}(\mu)$ states, alphabet size of $2 \cdot \mu - 1$, and transition-density of $\mathcal{O}(\mu^2)$.

5.3.2 QuIP: Algorithm description

The construction of the DS comparator automata in Section 5.3.1 leads to an implementation-friendly QuIP from BCV. The core focus of QuIP is to ensure that

Algorithm 3 QuIP(P, Q, d), Is $P \subseteq_d Q$?

1: **Input:** DS automata P and Q with integer discount factor d
2: **Output:** True if $P \subseteq_d Q$, False otherwise
3: $\hat{P} \leftarrow$ AugmentWtAndLabel(P)
4: $\hat{Q} \leftarrow$ AugmentWt(Q)
5: $\hat{P} \times \hat{Q} \leftarrow$ MakeProductSameAlpha$(\hat{P}, \hat{Q})$
6: $\mathcal{A} \leftarrow$ MakeBaseline$(\mu, d, \leq)$
7: $DomWithWitness \leftarrow$ IntersectSelectAlpha$(\hat{P} \times \hat{Q}, \mathcal{A})$
8: $Dom \leftarrow$ ProjectOutWt$(DomWithWitness)$
9: $\hat{P}_{-wt} \leftarrow$ ProjectOutWt$(\hat{P})$
10: **return** $\hat{P}_{-wt} \subseteq Dom$

the size of intermediate automata is small and they have fewer transitions to assist the LI solvers. Technically, QuIP differs from BCV by incorporating the new comparator automata and an appropriate Intersect function, rendering QuIP theoretical improvement over BCV. Like BCV, QuIP also determines all diminished runs of P. So, it disambiguates P by appending weight and a unique label to each of its transitions. Since, the identity of runs of Q is not important, we do not disambiguate between runs of Q, we only append the weight to each transition (Algorithm 3, Line 4). The DS comparator is constructed for discount factor d, maximum weight μ along transitions in P and Q, and the inequality $\leq$. Since the alphabet of the DS comparator are integers between $-\mu$ to μ, the alphabet of the product $\hat{P} \times \hat{\mathbb{Q}}$ is adjusted accordingly. Specifically, the weight recorded along transitions in the product is taken to be the difference of weight in $\hat{P}$ to that in $\hat{Q}$ i.e. if $\tau_P : s_1 \xrightarrow{a_1, wt_1, l} s_2$ and $\tau_Q : t_1 \xrightarrow{a_2, wt_2} t_2$ are transitions in $\hat{P}$ and $\hat{Q}$ respectively, then $\tau = (s_1, t_1) \xrightarrow{a_1, wt_1 - wt_2, l} (s_2, t_2)$ is a tran-

sition in $\hat{P} \times \hat{Q}$ iff $a_1 = a_2$ (Algorithm 3, Line 5). In this case, Intersect intersects the DS comparator $\mathcal{A}$ and product $\hat{P} \times \hat{Q}$ only on the weight-component of alphabet in $\hat{P} \times \hat{Q}$. Specifically, if $s_1 \xrightarrow{(a,wt_1,l)} s_2$ and $t_1 \xrightarrow{wt_2} t_2$ are transitions in $\hat{P} \times \hat{Q}$ and comparator $\mathcal{A}^{\mu,d}_{\leq}$ respectively, then $(s_1, t_1, i) \xrightarrow{a,wt_1,l} (s_2, t_2, j)$ is a transition in the intersection iff $wt_1 = wt_2$, where $j = (i + 1) \mod 2$ if either s_1 or t_1 is an accepting state, and $j = i$ otherwise. Automaton *Dom* and $\hat{P}_{-wt}$ are obtained by project out the weight-component from the alphabet of $\hat{P} \times \hat{Q}$ and $\hat{P}$ respectively. The alphabet of $\hat{P} \times \hat{Q}$ and $\hat{P}$ are converted from (a, wt, l) to only (a, l). It is necessary to project out the weight component since in $\hat{P} \times \hat{Q}$ they represent the difference of weights and and in $\hat{P}$ they represent the absolute value of weight.

Finally, the language of *Dom* is equated with that of $\hat{P}_{-wt}$ which is the automaton generated from $\hat{P}$ after discarding weights from transitions. However, it is easy to prove that $Dom \subseteq \hat{P}_{-wt}$. Therefore, instead of language-equivalence between *Dom* and $\hat{P}_{-wt}$ and, it is sufficient to check whether $\hat{P}_{-wt} \subseteq Dom$. As a result, QuIP utilizes LI solvers as a black-box to perform this final step.

Lemma 5.7 [Trans. Den. in QuIP] *Let δ_P, δ_Q denote transition-densities of P and Q, resp., and μ be the upper bound for DS comparator $\mathcal{B}^{\mu,d}_{\leq}$. Number of states and transition-density of Dom are $\mathcal{O}(s_P s_Q \mu)$ and $\mathcal{O}(\delta_P \delta_Q \cdot \mu^2)$, respectively.*

Theorem 5.5 [Practical complexity of QuIP] *Let P and Q be DS automata with s_P, s_Q number of states, respectively, and maximum weight on transitions be μ. Worst-case complexity for QuIP for integer discount-factor $d > 1$ when language-equivalence is performed via Ramsay-based inclusion testing is $2^{\mathcal{O}(s_P^2 \cdot s_Q^2 \cdot \mu^2)}$.*

Theorem 5.5 demonstrates that while complexity of QuIP (in practice) improves upon BCV (in practice), it is still worse than DetLP.

5.4 Empirical analysis of DS inclusion algorithms

We provide implementations of our tools QuIP and DetLP and conduct experiments on a large number of synthetically-generated benchmarks to compare their performance. We seek to find answers to the following questions:

1. Which tool has better performance, as measured by runtime, and number of benchmarks solved?
2. How does change in transition-density affect performance of the tools?
3. How dependent are our tools on their underlying solvers?

Implementation details

We implement our tools QuIP and DetLP in C++, with compiler optimization `o3` enabled. We implement our own library for all Büchi-automata and DS-automata operations, except for language-inclusion for which we use the state-of-the-art LI-solver RABIT [5] as a black-box. We enable the `-fast` flag in RABIT, and tune its JAVA-threads with Xss, Xms, Xmx set to 1GB, 1GB and 8GB respectively. We use the large-scale LP-solver GLPSOL provided by GLPK (GNU Linear Programming Kit) [2] inside DetLP. We did not tune GLPSOL since it consumes a very small percentage of total time in DetLP, as we see later in Fig 5.3b.

We also employ some implementation-level optimizations. Various steps of QuIP and DetLP such as product, DS-determinization, baseline construction, involve the creation of new automaton states and transitions. We reduce their size by adding a new state only if it is reachable from the initial state, and a new transition only if it originates from such a state.

The universal automata is constructed on the restricted alphabet of only those weights that appear in the product $\hat{P} \times \hat{Q}$ to include only necessary transitions. We also reduce its size with Büchi minimization tool Reduce [5].

Since all states of $\hat{P} \times \hat{Q}$ are accepting, we conduct the intersection so that it avoids doubling the number of product states. This can be done, since it is sufficient to keep track of whether words visit accepting states in the universal.

Benchmarks

To the best of our knowledge, there are no standardized benchmarks for DS-automata. We attempted to experimented with examples that appear in research papers. However, these examples are too few and too small, and do not render an informative view of performance of the tools. Following a standard approach to performance evaluation of automata-theoretic tools [6, 71, 86], we experiment with our tools on *randomly generated* benchmarks.

Random weighted-automata generation The parameters for our random weighted-automata generation procedure are the number of states N, transition-density δ and upper-bound μ for weight on transitions. The states are represented by the set $\{0, 1, \ldots, N-1\}$. All states of the weighted-automata are accepting, and they have a unique initial state 0. The alphabet for all weighted-automata is fixed to $\Sigma = \{a, b\}$. Weight on transitions ranges from 0 to $\mu - 1$. For our experiments we only generate complete weighted-automata. These weighted automata are generated only if the number of transitions $\lfloor N \cdot \delta \rfloor$ is greater than $N \cdot |\Sigma|$, since there must be at least one transition on each alphabet from every state. We first complete the weighted-automata by creating a transition from each state on every alphabet.

In this case the destination state and weight are chosen randomly. The remaining $(N \cdot |\Sigma| - \lfloor N \cdot \delta \rfloor)$-many transitions are generated by selecting all parameters randomly i.e. the source and destination states from $\{0, \ldots N-1\}$, the alphabet from Σ, and weight on transition from $\{0, \mu - 1\}$.

Design and setup for experimental evaluation

Our experiments were designed with the objective to compare DetLP and QuIP. Due to the lack of standardized benchmarks, we conduct our experiments on randomly-generated benchmarks. Therefore, the parameters for $P \subseteq_d Q$ are the number of states s_P and s_Q, transition density δ, and maximum weight wt. We seek to find answers to the questions described at the beginning of § 5.4.

Each instantiation of the parameter-tuple (s_P, s_Q, δ, wt) and a choice of tool between QuIP and DetLP corresponds to one experiment. In each experiment, the weighted-automata P and Q are randomly-generated with the parameters (s_P, δ, wt) and (s_Q, δ, wt), respectively, and language-inclusion is performed by the chosen tool. Since all inputs are randomly-generated, each experiment is repeated for 50 times to obtain statistically significant data. Each experiment is run for a total of 1000 sec on for a single node of a high-performance cluster. Each node of the cluster consists of two quad-core Intel-Xeon processor running at 2.83GHz, with 8GB of memory per node. The runtime of experiments that do not terminate within the given time limit is assigned a runtime of ∞. We report the median of the runtime-data collected from all iterations of the experiment.

These experiments are scaled-up by increasing the size of inputs. The worst-case analysis of QuIP demonstrates that it is symmetric in s_P and s_Q, making the algorithm impartial to which of the two inputs is scaled (Theorem 5.5). On the other hand,

complexity of DetLP is dominated by s_Q (Theorem 5.2). Therefore, we scale-up our experiments by increasing s_Q only.

Since DetLP is restricted to complete automata, these experiments are conducted on complete weighted automata only. We collect data on total runtime of each tool, the time consumed by the underlying solver, and the number of times each experiment terminates with the given resources. We experiment with $s_P = 10$, δ ranges between 2.5-4 in increments of 0.5 (we take lower-bound of 2.5 since $|\Sigma| = 2$), $wt \in \{4, 5\}$, and s_Q ranges from 0-1500 in increments of 25, $d = 3$. These sets of experiments also suffice for testing scalability of both tools.

Observations

We first compare the tools based on the number of benchmarks each can solve. We also attempt to unravel the main cause of failure of each tool. Out of the 50 experiments for each parameter-value, DetLP consistently solves more benchmarks than QuIP for the same parameter-values (Fig. 5.2a-5.2b)*. The figures also reveal that both tools solve more benchmarks at lower transition-density. The most common, in fact almost always, reason for QuIP to fail before its timeout was reported to be memory-overflow inside RABIT during language-inclusion between $\hat{P}_{-wt}$ and Dom. On the other hand, the main cause of failure of DetLP was reported to be memory overflow during DS-determinization and preprocessing of the determinized DS-automata before GLPSOL is invoked. This occurs due to the sheer size of the determinized DS-automata, which can very quickly become very large. These empirical observations indicate that the bottleneck in QuIP and DetLP may be language-inclusion and explicit DS-determinization, respectively.

*Figures are best viewed online and in color

We investigate the above intuition by analyzing the runtime trends for both tools. Fig. 5.3a plots the runtime for both tools. The plot shows that QuIP fares significantly better than DetLP in runtime at $\delta = 2.5$. The plots for both the tools on logscale seem curved (Fig. 5.3a), suggesting a sub-exponential runtime complexity. These were observed at higher δ as well. However, at higher δ we observe very few outliers on the runtime-trend graphs of QuIP at larger inputs when just a few more than 50% of the runs are successful. This is expected since effectively, the median reports the runtime of the slower runs in these cases. Fig 5.3b records the ratio of total time spent inside RABIT and GLPSOL . The plot reveals that QuIP spends most of its time inside RABIT. We also observe that most memory consumptions in QuIP occurs inside RABIT. In contrast, GLPSOL consumes a negligible amount of time and memory in DetLP. Clearly, performance of QuIP and DetLP is dominated by RABIT and explicit DS-determinization, respectively.

We also determined how runtime performance of tools changes with increasing discount-factor d. Both tools consume lesser time as d increases.

Finally, we test for scalability of both tools. In Fig. 5.4a, we plot the median of total runtime as s_Q increases at $\delta = 2.5, 3$ ($s_P = 10, \mu = 4$) for QuIP. We attempt to best-fit the data-points for each δ with functions that are linear, quadratic and cubic in s_Q using squares of residuals method. Fig 5.4b does the same for DetLP. We observe that QuIP and DetLP are best fit by functions that are linear and quadratic in s_Q, respectively.

Inferences and discussion

Our empirical analysis arrives at conclusions that a purely theoretical exploration would not have. First of all, we observe that despite having a the worse theoretical

complexity, the median-time complexity of QuIP is better than DetLP by an order of n. In theory, QuIP scales exponentially in s_Q, but only linearly in s_Q in runtime. Similarly, runtime of DetLP scales quadratically in s_Q. The huge margin of complexity difference emphasizes why solely theoretical analysis of algorithms is not sufficient.

Earlier empirical analysis of LI-solvers had made us aware of their dependence on transition-density δ. As a result, we were able to design QuIP cognizant of parameter δ. Therefore, its runtime dependence on δ is not surprising. However, our empirical analysis reveals runtime dependence of DetLP on δ. This is unexpected since δ does not appear in any complexity-theoretic analysis of DetLP (Theorem 5.1). We suspect this behavior occurs because the creation of each transition, say on alphabet a, during DS-determinization requires the procedure to analyze every transition on alphabet a in the original DS-automata. Higher the transition-density, more the transitions in the original DS-automata, hence more expensive is the creation of transitions during DS-determinization.

We have already noted that the performance of QuIP is dominated by RABIT in space and time. Currently, RABIT is implemented in Java. Although RABIT surpasses all other LI-solvers in overall performance, we believe it can be improved significantly via a more space-efficient implementation in a more performance-oriented language like C++. This would, in-turn, enhance QuIP.

The current implementation of DetLP utilizes the vanilla algorithm for DS-determinization. Since DS-determinization dominates DetLP, there is certainly merit in designing efficient algorithms for DS-determinization. However, we suspect this will be of limited advantage to DetLP since it will persist to incur the complete cost of explicit DS-determinization due to the separation of automata-theoretic and numeric reasoning.

Based on our observations, we propose to extract the complementary strengths of both tools: First, apply QuIP with a small timeout; Since DetLP solves more benchmarks, apply DetLP only if QuIP fails.

5.5 Chapter summary

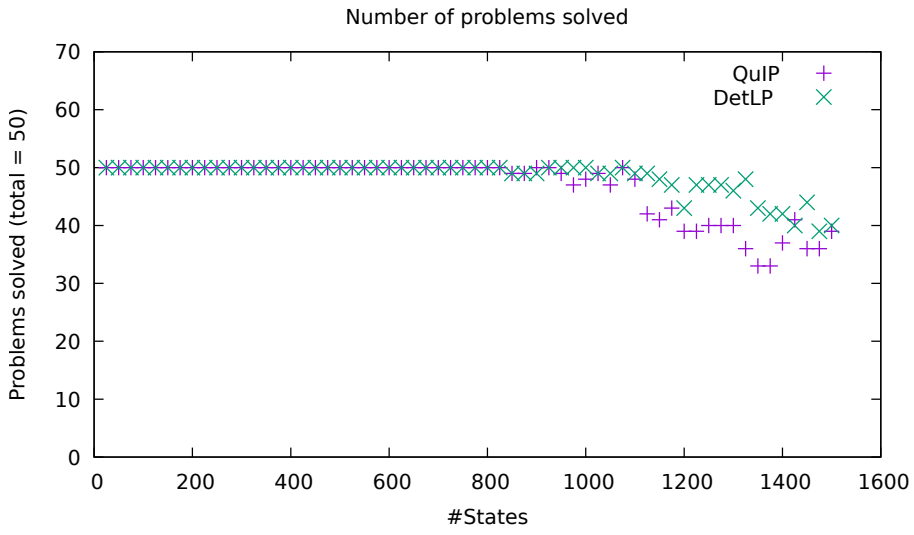
This chapter presents the first empirical evaluation of algorithms and tools for DS inclusion. We present two tools DetLP and QuIP. Our first tool DetLP is based on explicit DS-determinization and linear programming, and renders an exponential time and space algorithm. Our second tool QuIP improves upon a previously known comparator-based automata-theoretic algorithm BCV by means of an optimized comparator construction. Despite its PSPACE-complete theoretical complexity, we note that all practical implementations of QuIP are also exponential in time and space.

The focus of this work is to investigate these tools in practice. In theory, the exponential complexity of QuIP is worse than DetLP. Our empirical evaluation reveals the opposite: The median-time complexity of QuIP is better than DetLP by an order of n. Specifically, QuIP scales linearly while DetLP scales quadratically in the size of inputs. This re-asserts the gap between theory and practice, and aserts the need of better metrics for practical algorithms. Further emprirical analysis by by scaling the right-hand side automaton will be beneficial.

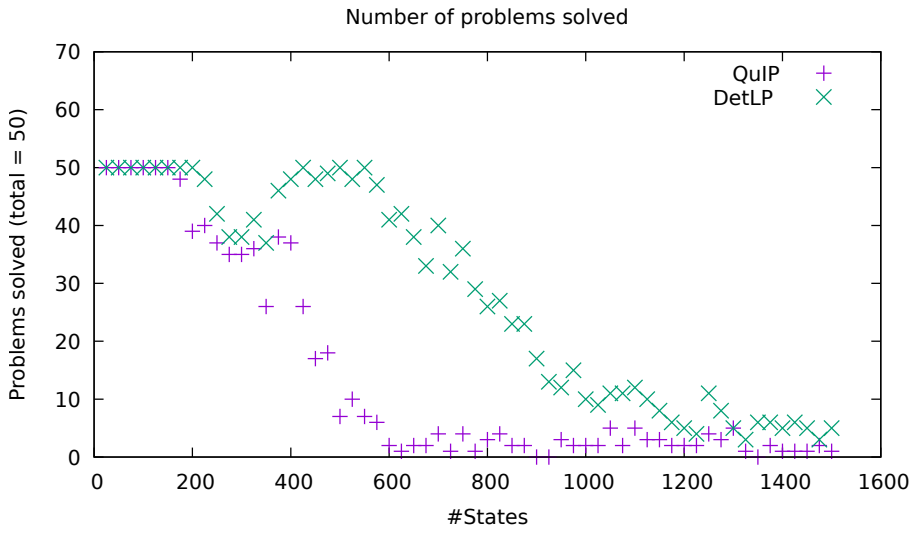
Nevertheless, DetLP consistently solves more benchmarks than QuIP. Most of QuIP's experiments fail due to memory-overflow within the LI-solver, indicating that more space-efficient implementations of LI-solvers would boost QuIP's performance. We are less optimistic about DetLP though. Our evaluation highlights the impediment of explicit DS-determinization, a cost that is unavoidable in DetLP's separation-of-techniques approach. This motivates future research that integrates automata-

theoretic and numerical reasoning by perhaps combining implicit DS-determinzation with baseline automata-like reasoning to design an on-the-fly algorithm for DS-inclusion.

Last but not the least, our empirical evaluations lead to discovering dependence of runtime of algorithms on parameters that had not featured in their worst-case theoretical analysis, such as the dependence of DetLP on transition-density. Such evaluations build deeper understanding of algorithms, and will hopefully serve a guiding light for theoretical and empirical investigation in-tandem of algorithms for quantitative analysis

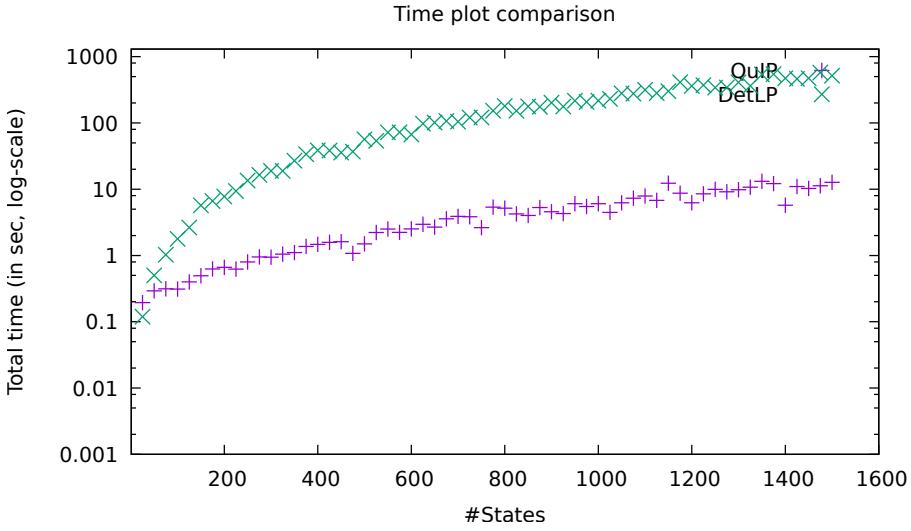


(a)

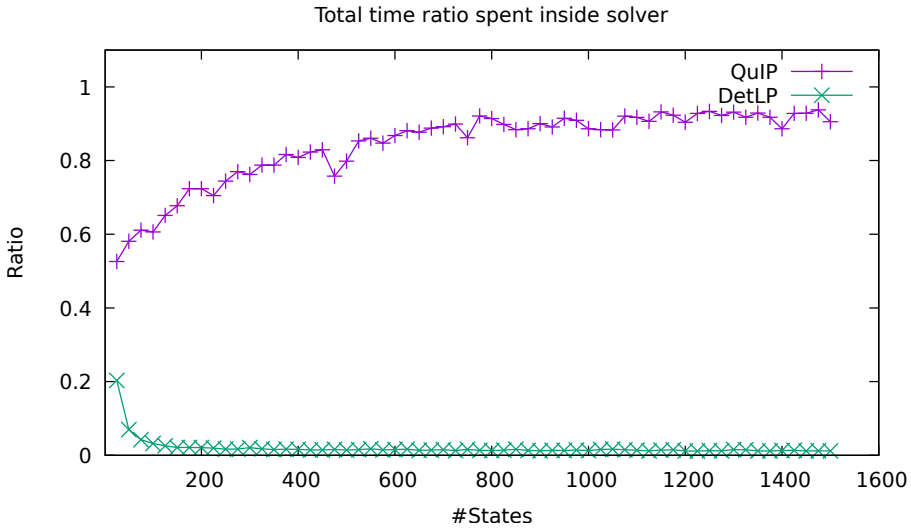


(b)

Figure 5.2 : Number of benchmarks solved by QuIP and DetLP out of 50 as s_Q increases with $s_P = 10$, $\mu = 4$. $\delta = 2.5$ and $\delta = 4$ in Fig 5.2a and Fig 5.2b, respectively.

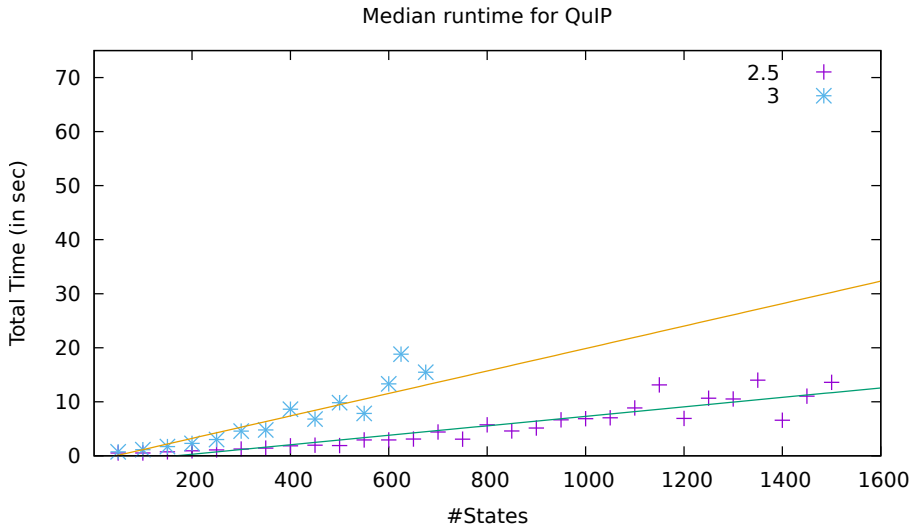


(a)

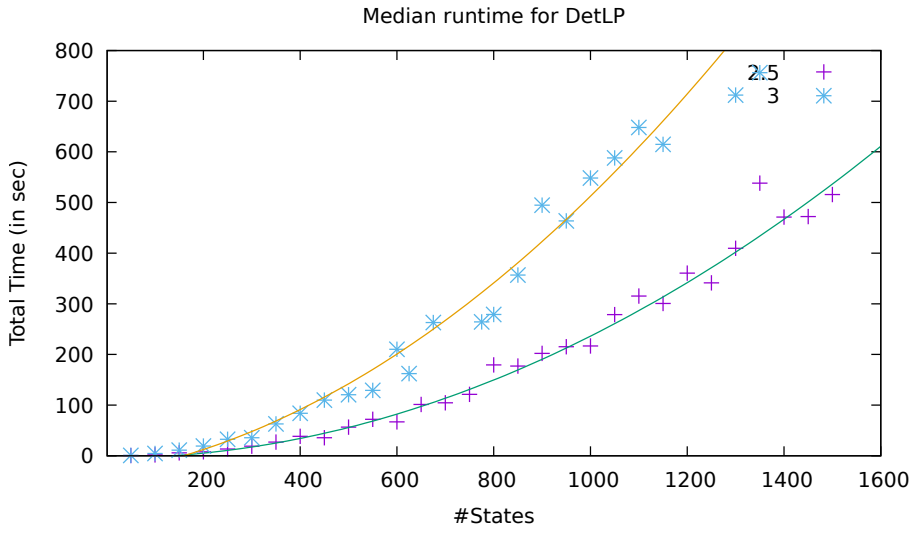


(b)

Figure 5.3 : Runtime performance trends of QuIP and DetLP: Fig 5.3a plots total runtime as s_Q increases $s_P = 10, \mu = 4$, $\delta = 2.5$. Figure shows median-time for each parameter-value. Fig 5.3b plots the ratio of time spent by tool inside its solver at the same parameter values.



(a)



(b)

Figure 5.4 : Scalability of QuIP (Fig 5.4a) and DetLP (Fig 5.4b) at $\delta = 2.5, 3$. Figures show median-time for each parameter-value.

Chapter 6

Safety/co-safety comparators for DS inclusion

Despite being an integrated method, the comparator-based algorithm for DS inclusion with integer discount factors does not outperform separation-of-techniques on all accounts. In particular, despite exhibiting improved runtime they did not manage to solve as many benchmarks as separation-of-techniques. This chapter aims at improving the comparator-based method for DS inclusion by diving deeper into language-theoretic properties of DS comparison languages and DS comparators. To this end, we explore safety and co-safety properties of comparison languages, and demonstrate comparator-based solutions are able to mitigate their weaknesses by leveraging the benefits of safety/co-safety languages/automata.

6.1 The predicament of Büchi complementation

In theory, DS inclusion for integer discount factors is PSPACE-complete [25]. Recent algorithmic approaches have tapped into language-theoretic properties of discounted-sum aggregate function [25, 42] to design practical algorithms for DS inclusion [24, 25]. These algorithms use *DS comparator automata* (*DS comparator*, in short) as their main technique, and are *purely* automata-theoretic. While these algorithms outperform other existing approaches for DS inclusion in runtime [31, 39], even these do not scale well on weighted-automata with more than few hundreds of states [24].

An in-depth examination of the DS comparator based algorithm exposes their scal-

ability bottleneck. DS comparator is a Büchi automaton that relates the discounted-sum aggregate of two (bounded) weight-sequences A and B by determining the membership of the interleaved pair of sequences (A, B) in the language of the comparator. As a result, DS comparators reduce DS inclusion to language inclusion between (nondeterministic) Büchi automaton. In spite of the fact that many techniques have been proposed to solve Büchi language inclusion efficiently in practice [7, 51], none of them can avoid at least an exponential blow-up of $2^{\mathcal{O}(n \log n)}$, for an n-sized input, caused by a direct or indirect involvement of Büchi complementation [80, 88].

This work meets the scalability challenge of DS inclusion by eradicating the dependence on Büchi language inclusion and Büchi complementation. We discover that DS comparators can be expressed by specialized Büchi automata called *safety automata* or *co-safety automata* [63] (§ 6.4.1). Safety and co-safety automata have the property that their complementation is performed by simpler and lower $2^{\mathcal{O}(n)}$-complexity subset-construction methods [64]. As a result, they facilitate a procedure for DS inclusion that uses subset-construction based intermediate steps instead of Büchi complementation, yielding an improvement in theoretical complexity from $2^{\mathcal{O}(n \cdot \log n)}$ to $2^{\mathcal{O}(n)}$. Our subset-construction based procedure has yet another advantage over Büchi complementation as they support efficient on-the-fly implementations, yielding practical scalability as well (§ 6.5).

An empirical evaluation of our prototype tool QuIPFly for the proposed procedure against the prior DS-comparator algorithm and other existing approaches for DS inclusion shows that QuIPFly outperforms them by orders of magnitude both in runtime and the number of benchmarks solved (§ 6.5).

Therefore, this chapter meets the scalability challenge of DS inclusion by delving deeper into language-theoretic properties of discounted-sum aggregate functions. By

doing so, we obtain algorithms for DS inclusion that render both tighter theoretical complexity and improved scalability in practice. Hence, we can conclude that comparator-based methods are effective in practice as well.

Organization This chapter goes as follows. Section 6.2 reviews classes of Büchi automata for which complementation can be performed without using Safra's complementation method [80]. Section 6.3 introduces safety/co-safety comparison languages and correspondingly their safety/co-safety comparator automata. We prove that quantitative inclusion performed with safety/co-safety comparators is more efficient than quantitative inclusion with ω-regular comparators. Section 6.4 studies DS aggregation in depth, and shows that DS comparion languages are either safety/co-safety languages. We use this newly discovered property to design much compact and deterministic constructions for DS comparators, and finally utilize these comparators to re-design DS inclusion. Finally, Section 6.5 conducts the empirical evaluation, and affirms the superiority of comparator-based methods for DS inclusion with integer discount factors.

6.2 Safraless automata

We coin the terms *Safraless automata* to refer to Büchi automata for which complementation can be performed without Safra's Büchi complementation [80]. We review two classes of Safraless automata [63]: (a). Safety and co-safety automata, and (b). Weak Büchi automata. For both of these classes of Safraless automata, complementation can be performed using simpler methods that use subset-construction. As a result, these automata incur a $2^{\mathcal{O}(n)}$ blow-up as opposed to a $2^{\mathcal{O}(n \log n)}$ blow-up, where n is the number of states in the original automaton. In addition, subset-construction

based methods are more amenable to efficient practical implementations since they can be performed on-the-fly.

6.2.1 Safety and co-safety automata

Let $\mathcal{L} \subseteq \Sigma^\omega$ be a language over alphabet Σ. A finite word $w \in \Sigma^*$ is a *bad prefix* for $\mathcal{L}$ if for all infinite words $y \in \Sigma^\omega$, $x \cdot y \notin \mathcal{L}$. A language $\mathcal{L}$ is a *safety language* if every word $w \notin \mathcal{L}$ has a bad prefix for $\mathcal{L}$. A language $\mathcal{L}$ is a *co-safety language* if its complement language is a safety language [13]. When a safety or co-safety language is an ω-regular language, the Büchi automaton representing it is called a safety or co-safety automaton, respectively [63]. Wlog, safety and co-safety automaton contain a *sink state* from which every outgoing transitions loops back to the sink state and there is a transition on every alphabet symbol. All states except the sink state are accepting in a safety automaton, while only the sink state is accepting in a co-safety automaton. The complementation of safety automaton is a co-safety automaton, and vice-versa. Safety automata are closed under intersection, and co-safety automata are closed under union.

6.2.2 Weak Büchi automaton

A Büchi automaton $\mathcal{A} = (S, \Sigma, \delta, s_I, \mathcal{F})$ is *weak* [64] if its states S can be partitioned into disjoint sets S_i, such that (a). For each set S_i, either $S_i \subseteq \mathcal{F}$ or $S_i \cap \mathcal{F} = \emptyset$ i.e. partitions are either accepting or rejecting, and (b). There exists a partial order $\leq$ on the collection of the S_i such that for every $s \in S_i$ and $t \in S_j$ for which $t \in \delta(s, a)$ for some $a \in \Sigma$, $S_j \leq S_i$ i.e. The transition function is restricted so that in each transition, the automaton either stays at the same set or moves to a set smaller in the partial order.

It is worth noting that safety and co-safety automata are also weak Büchi automata. In both of these cases, the states of the regular safety/co-safety automata are partitioned into two disjoint sets: The first set contains all but the sink state, and the second set contains the sink state only. Note how in both cases, the sets have either all accepting states or all non-accepting states, and the transitions are unidirectional order w.r.t to the partitions.

We make the observation that the intersection of a safety and co-safety automaton is indeed a weak Büchi automata. We prove it as follows:

Theorem 6.1 *The intersection of a safety automata and a co-safety automata is a weak Büchi automata.*

Proof 33 Wlog, let the two partitions in the co-safety automaton be S_C and $\{\mathsf{sink}_C\}$, and the partitions in the safety automaton be S_S and $\{\mathsf{sink}_S\}$. Their accepting partitions are $\{\mathsf{sink}_C\}$ and S_S, respectively.

We construct the intersection by taking simple product of both automata without the cyclic counter. The product automaton is a weak Büchi automaton, with four partitions obtained from all combinations of partitions of the co-safety and safety automata. Transitions from partition (S_C,S_S) could go to all four partitions. This partition also contains the initial state. Transitions from partitions (S_C,$\{\mathsf{sink}_S\}$) or ($\{\mathsf{sink}_C\}$,S_S) either go to themselves or to the sink partition ($\{\mathsf{sink}_C\}, \{\mathsf{sink}_S\}$). There are no transitions between (S_C,$\{\mathsf{sink}_S\}$) and ($\{\mathsf{sink}_C\}$,S_S).

Based on the transition relation, we assign the order of partitions as follows:

- Partition (S_C,S_S) has highest order
- Parition ($\{\mathsf{sink}_C\}, \{\mathsf{sink}_S\}$) has lowest order

- Other two partitions take any order in between, since there are no transitions in between these paritions.

Clearly, the partition (S_C,{sink_S}) is the only accepting partition. ■

6.3 Safety and co-safety comparison languages

We introduce comparator automata that are safety and co-safety automata Section 6.3.1. Section 6.3.2 designs an algorithm for quantitative inclusion with regular safety/co-safety comparators. We observe that the complexity of quantitative inclusion with safety/co-safety comparator automata is complexity is lower than that with ω-regular comparators.

6.3.1 Safety and co-safety comparison languages and comparators

We begin with formal definitions of safety/co-safety comparison languages and safety/co-safety comparators:

Definition 6.1 (Safety and co-safety comparison languages) Let Σ be a finite set of integers, $f : \mathbb{Z}^\omega \to \mathbb{R}$ be an aggregate function, and $\mathsf{R} \in \{\leq, <, \geq, >, =, \neq\}$ be a relation. A comparison language L over $\Sigma \times \Sigma$ for aggregate function f and relation R is said to be a safety comparison language (or a co-safety comparison language) if L is a safety language (or a co-safety language).

Definition 6.2 (Safety and co-safety comparators) Let Σ be a finite set of integers, $f : \mathbb{Z}^\omega \to \mathbb{R}$ be an aggregate function, and $\mathsf{R} \in \{\leq, <, \geq, >, =, \neq\}$ be a relation. A comparator for aggregate function f and relation R is a *safety comparator* (or *co-safety comparator*) is the comparison language for f and R is a safety language (or co-safety language).

A safety comparator is *regular* if its language is ω-regular (equivalently, if its automaton is a safety automaton). Likewise, a co-safety comparator is *regular* if its language is ω-regular (equivalently, automaton is a co-safety automaton).

Just like the closure property of ω-regular comparison languages, safety and co-safety comparison languages exhibit closure under safety or co-safety languages as well. By complementation duality of safety and co-safety languages, comparison language for an aggregate function f for non-strict inequality $\leq$ is safety iff the comparison language for f for strict inequality $<$ is co-safety. Since safety languages and safety automata are closed under intersection, safety comparison languages and regular safety comparator for non-strict inequality renders the same for equality. Similarly, since co-safety languages and co-safety automata are closed under union, co-safety comparison languages and regular co-safety comparators for non-strict inequality render the same for the inequality relation. As a result, if the comparison language for any one relation is either a safety or co-safety language, the comparison languages for all relations will also be safety or co-safety languages. Therefore, it suffices to examine the comparison language for one relation only.

6.3.2 Quantitative inclusion with regular safety/co-safety comparators

This section covers the first technical contributions of this work. It studies safety and co-safety properties of comparison languages and comparators, and utilizes them to obtain tighter theoretical upper-bound for quantitative inclusion with these comparators that exhibit safety/co-safety and ω-regular properties as opposed to comparators that are only ω-regular.

Key Ideas

A run of word w in a weighted-automaton is *maximal* if its weight is the supremum weight of all runs of w in the weighted-automaton. A run ρ_P of w in P is a *counterexample* for $P \subseteq Q$ (or $P \subset Q$) iff there exists a maximal run sup_Q of w in Q such that $wt(\rho_P) > wt(sup_Q)$ (or $wt(\rho_P) \geq wt(sup_Q)$). Consequently, $P \subseteq Q$ (or $P \subset Q$) iff there are no counterexample runs in P. Therefore, the roadmap to solve quantitative inclusion for regular safety/co-safety comparators is as follows:

1. Use regular safety/co-safety comparators to construct the *maximal automaton* of Q i.e. an automaton that accepts all maximal runs of Q (Corollary 6.1).

2. Use the regular safety/co-safety comparator and the maximal automaton to construct a *counterexample automaton* that accepts all counterexample runs of the inclusion problem $P \subseteq Q$ (or $P \subset Q$) (Lemma 6.3).

3. Solve quantitative inclusion for safety/co-safety comparator by checking for emptiness of the counterexample (Theorem 6.2).

Let W be a weighted automaton. Then the *annotated automaton* of W, denoted by $\hat{W}$, is the Büchi automaton obtained by transforming transition $s \xrightarrow{a} t$ with weight v in W to transition $s \xrightarrow{a,v} t$ in $\hat{W}$. Observe that $\hat{W}$ is a safety automaton since all its states are accepting. A run on word w with weight sequence wt in W corresponds to an *annotated word* (w, wt) in $\hat{W}$, and vice-versa.

Maximal automaton

This section covers the construction of the *maximal automaton* from a weighted automaton. Let W and $\hat{W}$ be a weighted automaton and its annotated automaton,

respectively. We call an annotated word (w, wt_1) in $\hat{W}$ *maximal* if for all other words of the form (w, wt_2) in $\hat{W}$, $wt(wt_1) \geq wt(wt_2)$. Clearly, (w, wt_1) is a maximal word in $\hat{W}$ iff word w has a run with weight sequence wt_1 in W that is maximal. We define *maximal automaton* of weighted automaton W, denoted $\mathsf{Maximal}(W)$, to be the automaton that accepts all maximal words of its annotated automata $\hat{W}$.

We show that when the comparator is regular safety/co-safety, the construction of the maximal automata incurs a $2^{\mathcal{O}(n)}$ blow-up. This section exposes the construction for maximal automaton when comparator for non-strict inequality is regular safety. The other case when the comparator for strict inequality is regular co-safety is similar, hence skipped.

Lemma 6.1 *Let W be a weighted automaton with regular safety comparator for non-strict inequality. Then the language of* $\mathsf{Maximal}(W)$ *is a safety language.*

Proof 34 Intuitively, an annotated word (w, wt_1) is not maximal in $\hat{W}$ for one of the following two reasons: Either (w, wt_1) is not a word in $\hat{W}$, or there exists another word (w, wt_2) in $\hat{W}$ s.t. $wt(wt_1) < wt(wt_2)$ (equivalently (wt_1, wt_2) is not in the comparator non-strict inequality). Both $\hat{W}$ and comparator for non-strict inequality are safety languages, so the language of maximal words must also be a safety language.

In detail, we show that every annotated word that is not a maximal word in $\hat{W}$ must have a bad-prefix. An annotated word (w, wt_1) is not maximal in $\hat{W}$ for one of two reasons: Either (w, wt_1) is not a word in $\hat{W}$, or there exists another word (w, wt_2) in $\hat{W}$ s.t. $wt(wt_1) < wt(wt_2)$.

If (w, wt_1) is not a word in $\hat{W}$, since $\hat{W}$ is a safety automata as well, (w, wt_1) must have a bad-prefix w.r.t $\hat{W}$. The same bad-prefix serves as a bad-prefix w.r.t. the maximal language since none of its extensions are in $\hat{W}$ either.

Otherwise since the comparator for $<$ is ω-regular co-safety, there exists an n-length prefix of (wt_1, wt_2) s.t. all extensions of $(wt_1, wt_2)[n]$ are present in the ω-regular co-safety automaton. We claim the prefix of (w, wt_1) of same length n is a bad-prefix for the maximal language. Consider the n-length prefix $(w, wt_1)[n]$ of (w, wt_1). Let (w_{ext}, wt_{ext}) be an infinite extension of $(w, wt)[n]$. If (w_{ext}, wt_{ext}) is not a word in $\hat{W}$, it is also not a maximal word in $\hat{W}$.

If (w_{ext}, wt_{ext}) is a word in $\hat{W}$, Let (w_{ext}, wt'_{ext}) be and extensions of $(w, wt_2)[n]$. Note that the first component of extension of $(w, wt_1)[n]$ and $(w, wt_2)[n]$ are the same. This is possible since the underlying Büchi automata of W is a complete automaton. Since (wt_{ext}, wt'_{ext}) is an extension of $(wt_1, wt_2)[n]$, $wt(wt_{ext}) < wt(wt'_{ext})$. Therefore, (w_{ext}, wt_{ext}) is not a maximal word. ■

We now proceed to construct the safety automata for $\mathsf{Maximal}(W)$

Intuition The intuition behind the construction of maximal automaton follows directly from the definition of maximal words. Let $\hat{W}$ be the annotated automaton for weighted automaton W. Let $\hat{\Sigma}$ denote the alphabet of $\hat{W}$. Then an annotated word $(w, wt_1) \in \hat{\Sigma}^\omega$ is a word in $\mathsf{Maximal}(W)$ if (a) $(w, wt_1) \in \hat{W}$, and (b) For all words $(w, wt_2) \in \hat{W}$, $wt(wt_1) \geq wt(wt_2)$.

The challenge here is to construct an automaton for condition (b). Intuitively, this automaton simulates the following action: As the automaton reads word (w, wt_1), it must spawn all words of the form (w, wt_2) in $\hat{W}$, while also ensuring that $wt(wt_1) \geq wt(wt_2)$ holds for every word (w, wt_2) in $\hat{W}$. Since $\hat{W}$ is a safety automaton, for a word $(w, wt_1) \in \hat{\Sigma}^\omega$, all words of the form $(w, wt_2) \in \hat{W}$ can be traced by subset-construction. Similarly since the comparator C for non-strict inequality ($\geq$) is a safety automaton, all words of the form $(wt_1, wt_2) \in C$ can be traced by subset-

construction as well. The construction needs to carefully align the word (w, wt_1) with the all possible $(w, wt_2) \in \hat{W}$ and their respective weight sequences $(wt_1, wt_2) \in C$.

Construction and detailed proof Let W be a weighted automaton, with annotated automaton $\hat{W}$ and C denote its regular safety comparator for non-strict inequality. Let S_W denote the set of states of W (and $\hat{W}$) and S_C denote the set of states of C. We define $\mathsf{Maximal}(W) = (S, s_I, \hat{\Sigma}, \delta, \mathcal{F})$ as follows:

- Set of states S consists of tuples of the form (s, X), where $s \in S_W$, and $X = \{(t, c) | t \in S_W, c \in S_C\}$
- $\hat{\Sigma}$ is the alphabet of $\hat{W}$
- Initial state $s_I = (s_w, \{(s_w, s_c)\})$, where s_w and s_c are initial states in $\hat{W}$ and C, respectively.
- Let states $(s, X), (s, X') \in S$ such that $X = \{(t_1, c_1), \ldots, (t_n, c_n)\}$ and $X' = \{(t'_1, c'_1), \ldots, (t'_m, c'_m)\}$. Then $(s, X) \xrightarrow{(a,v)} (s', X') \in \delta$ iff
  1. $s \xrightarrow{(a,v)} s'$ is a transition in $\hat{W}$, and
  2. $(t'_j, c'_j) \in X'$ if there exists $(t_i, c_i) \in X$, and a weight v' such that $t_i \xrightarrow{a,v'} t'_j$ and $c_i \xrightarrow{v,v'} c'_j$ are transitions in $\hat{W}$ and C, respectively.
- $(s, \{(t_1, c_1), \ldots, (t_n, c_n)\}) \in \mathcal{F}$ iff s and all t_i are accepting in $\hat{W}$, and all c_i is accepting in C.

Lemma 6.2 *Let W be a weighted automaton with regular safety comparator C for non-strict inequality. Then the size of* $\mathsf{Maximal}(W)$ *is $|W| \cdot 2^{\mathcal{O}(|W| \cdot |C|)}$.*

Proof 35 First, we prove that the automaton constructed above is a safety automaton. To do so, we prove that all outgoing transitions from an accepting states go into an accepting state. A state $(s, \{(t_1, c_1), \ldots, (t_n, c_n)\})$ is non-accepting in the automata if one of s,t_i or c_j is non-accepting in underlying automata $\hat{W}$ and the comparator. Since $\hat{W}$ and the comparator automata are safety, all outgoing transitions from a non-accepting state go to non-accepting state in the underlying automata. Therefore, all outgoing transitions from a non-accepting state in $\mathsf{Maximal}(W)$ go to non-accepting state in $\mathsf{Maximal}(W)$. Therefore, $\mathsf{Maximal}(W)$ is a safety automaton.

An analysis on the number of possible states of the form (s, X) is enough to see why the number of states is $|W| \cdot 2^{\mathcal{O}(|W|\cdot|C|)}$. This is because there are $|W|$ number of possibilities for s and $2^{\mathcal{O}(|W|\cdot|C|)}$ number of possibilities for X. ■

A similar construction proves that the maximal automata of weighted automata W with regular safety comparator C for strict inequality contains $|W| \cdot 2^{\mathcal{O}(|W|\cdot|C|)}$ states. The difference is that in this case the maximal automaton may not be a safety automaton. But it will still be a weak Büchi automaton. Therefore, Lemma 6.2 generalizes to:

Corollary 6.1 *Let W be a weighted automaton with regular safety/co-safety comparator C. Then* $\mathsf{Maximal}(W)$ *is either a safety automaton or a weak Büchi automaton of size* $|W| \cdot 2^{\mathcal{O}(|W|\cdot|C|)}$.

Counterexample automaton

This section covers the construction of the counterexample automaton. Given weighted automata P and Q, an annotated word (w, wt_P) in annotated automata $\hat{P}$ is a *counterexample word* of $P \subseteq Q$ (or $P \subset Q$) if there exists (w, wt_Q) in $\mathsf{Maximal}(Q)$ s.t.

$wt(wt_P) > wt(wt_Q)$ (or $wt(wt_P) \geq wt(wt_Q)$). Clearly, annotated word (w, wt_P) is a counterexample word iff there exists a counterexample run of w with weight-sequence wt_P in P.

For this section, we abbreviate strict and non-strict to strct and nstrct, respectively. For inc $\in \{$strct, nstrct$\}$, the *counterexample automaton* for inc-quantitative inclusion, denoted by Counterexample(inc), is the automaton that contains all counterexample words of the problem instance. We construct the counterexample automaton as follows:

Lemma 6.3 *Let P, Q be weighted-automata with regular safety/co-safety comparators. For* inc $\in \{$strct, nstrct$\}$, Counterexample(inc) *is a Büchi automaton.*

Proof 36 We construct Büchi automaton Counterexample(inc) for inc $\in \{$strct, nstrct$\}$ that contains the counterexample words of inc-quantitative inclusion. Since the comparator are regular safety/co-safety, Maximal(Q) is a Büchi automaton (Corollary 6.1). Construct the product $\hat{P} \times$ Maximal(Q) such that transition $(p_1, q_1) \xrightarrow{a,v_1,v_2} (p_1, q_2)$ is in the product iff $p_1 \xrightarrow{a,v_1} p_1$ and $q_1 \xrightarrow{a,v_2} q_2$ are transitions in $\hat{P}$ and Maximal(Q), respectively. A state (p, q) is accepting if both p and q are accepting in $\hat{P}$ and Maximal(Q). One can show that the product accepts (w, wt_P, wt_Q) iff (w, wt_P) and (w, wt_Q) are words in $\hat{P}$ and Maximal(Q), respectively.

If inc = strct, intersect $\hat{P} \times$ Maximal(Q) with comparator for $\geq$. If inc = nstrct, intersect $\hat{P} \times$ Maximal(Q) with comparator for $>$. Since the comparator is a safety or co-safety automaton, the intersection is taken without the cyclic counter. Therefore, $(s_1, t_1) \xrightarrow{a,v_1,v_2} (s_2, t_2)$ is a transition in the intersection iff $s_1 \xrightarrow{a,v_1,v_2} s_2$ and $t_1 \xrightarrow{v_1,v_2} t_2$ are transitions in the product and the appropriate comparator, respectively. State (s, t) is accepting if both s and t are accepting. The intersection will

accept (w, wt_P, wt_Q) iff (w, wt_P) is a counterexample of inc-quantitative inclusion. Counterexample(inc) is obtained by projecting out the intersection as follows: Transition $m \xrightarrow{a,v_1,v_2} n$ is transformed to $m \xrightarrow{a,v_1} n$. ■

Quantitative inclusion

In this section, we give the final algorithm for quantitative inclusion with regular safety/co-safety comparators, and contrast the worst-case complexity of quantitative inclusion with ω-regular and regular safety/co-safety comparators.

Theorem 6.2 *Let P, Q be weighted-automata with regular safety/co-safety comparators. Let $C_{\leq}$ and $C_{<}$ be the comparators for $\leq$ and $<$, respectively. Then*

- *Strict quantitative inclusion $P \subset Q$ is reduced to emptiness checking of a Büchi automaton of size $|P||C_{\leq}||Q| \cdot 2^{\mathcal{O}(|Q| \cdot |C_{<}|)}$.*
- *Non-strict quantitative inclusion $P \subseteq Q$ is reduced to emptiness checking of a Büchi automaton of size $|P||C_{<}||Q| \cdot 2^{\mathcal{O}(|Q| \cdot |C_{<}|)}$.*

Proof 37 Strict and non-strict are abbreviated to strct and nstrct, respectively. For inc $\in$ {strct, nstrct}, inc-quantitative inclusion holds iff Counterexample(inc) is empty. Size of Counterexample(inc) is the product of size of P, Maximal(Q) (Corollary 6.1), and the appropriate comparator as described in Lemma 6.3. ■

In contrast, quantitative inclusion with ω-regular comparators reduces to emptiness of a Büchi automaton with $|P| \cdot 2^{\mathcal{O}(|P||Q||C| \cdot \log(|P||Q||C|))}$ states [25]. The $2^{\mathcal{O}(n \log n)}$ blow-up is unavoidable due to Büchi complementation. Clearly, quantitative inclusion with regular safety/co-safety has lower worst-case complexity.

6.4 DS inclusion with safety/co-safety comparators

This section covers the second technical contributions of this chapter. It uncovers that DS comparison languages are safety/co-safety, and utilizes this information to obtain tighter theoretical upper-bound for DS inclusion when the discount factor is an integer. Unless mentioned otherwise, the discount-factor is an integer.

In § 6.4.1 we prove that DS comparison languages are either safety or co-safety for all rational discount-factors. Since DS comparison languages are ω-regular for integer discount-factors [25], we obtain that DS comparators for integer discount-factors form safety or co-safety automata. Next, § 6.4.2 makes use of newly obtained safety/co-safety properties of DS comparator to present the first deterministic constructions for DS comparators. These deterministic construction are compact in the sense that they are the present in their minimal automata form. Finally, since DS comparators are regular safety/co-safety, our analysis shows that the complexity of DS inclusion is improved as a consequence of the complexity observed for quantitative inclusion with regular safety/co-safety comparators.

Recall the definitions of DS comparison languages and DS comparators from Definition 5.1 and Definition 5.2, respectively. Here *DS comparison language* with upper bound μ, discount-factor $d > 1$, and relation R is the language that accepts an infinite-length and bounded weight sequence A over the alphabet $\{-\mu, \ldots, \mu\}$ iff $DS(A, d)$ R 0 holds. Similarly, DS comparator with the same parameters μ, $d > 1$, accepts the DS comparison language with parameters μ, d and R. We adopt these definitions in the Section 6.4.

A note on notation: Throughout this section, the concatenation of finite sequence x with finite or infinite sequence y is denoted by $x \cdot y$ in the following.

6.4.1 DS comparison languages and their safety/co-safety properties

The central result of this section is that DS comparison languages are safety or co-safety languages for all (integer and non-integer) discount-factors (Theorem 6.3). In particular, since DS comparison languages are ω-regular for integer discount-factors [25], this implies that DS comparators for integer discount-factors form safety or co-safety automata (Corollary 6.2). The argument for safety/co-safety of DS comparison languages depends on the property that the discounted-sum aggregate of all bounded weight-sequences exists for all discount-factors $d > 1$ [79].

Theorem 6.3 *Let $\mu > 1$ be the upper bound. For rational discount-factor $d > 1$*

1. *DS-comparison languages are safety languages for relations $\mathsf{R} \in \{\leq, \geq, =\}$*

2. *DS-comparison language are co-safety languages for relations $\mathsf{R} \in \{<, >, \neq\}$.*

Proof 38 Due to duality of safety/co-safety languages, it is sufficient to show that DS-comparison language with $\leq$ is a safety language.

Let us assume that DS-comparison language with $\leq$ is not a safety language. Let W be a weight-sequence in the complement of DS-comparison language with $\leq$ such that it does not have a bad prefix.

Since W is in the complement of DS-comparison language with $\leq$, $DS(W, d) > 0$. By assumption, every i-length prefix $W[i]$ of W can be extended to a bounded weight-sequence $W[i] \cdot Y^i$ such that $DS(W[i] \cdot Y^i, d) \leq 0$.

Note that $DS(W, d) = DS(W[i], d) + \frac{1}{d^i} \cdot DS(W[i \ldots], d)$, and $DS(W[i] \cdot Y^i, d) = DS(W[i], d) + \frac{1}{d^i} \cdot DS(Y^i, d)$. The contribution of tail sequences $W[i \ldots]$ and Y^i to the discounted-sum of W and $W[i] \cdot Y^i$, respectively diminishes exponentially as the value of i increases. In addition, since and W and $W[i] \cdot Y^i$ share a common i-length

prefix $W[i]$, their discounted-sum values must converge to each other. The discounted sum of W is fixed and greater than 0, due to convergence there must be a $k \geq 0$ such that $DS(W[k] \cdot Y^k, d) > 0$. Contradiction. Therefore, DS-comparison language with $\leq$ is a safety language.

The above intuition is formalized as follows: Since $DS(W, d) > 0$ and $DS(W[i] \cdot Y^i, d) \leq 0$, the difference $DS(W, d) - DS(W[i] \cdot Y^i, d) > 0$.

By expansion of each term, we get $DS(W, d) - DS(W[i] \cdot Y^i, d) = \frac{1}{d^i}(DS(W[i \ldots], d) - DS(Y^i, d)) \leq \frac{1}{d^i} \cdot (\mod DS(W[i \ldots], d) + \mod DS(Y^i, d))$. Since the maximum value of discounted-sum of sequences bounded by μ is $\frac{\mu \cdot d}{d-1}$, we also get that $DS(W, d) - DS(W[i] \cdot Y^i, d) \leq 2 \cdot \frac{1}{d^i} \mod \frac{\mu \cdot d}{d-1}$.

Putting it all together, for all $i \geq 0$ we get

$$0 < DS(W, d) - DS(W[i] \cdot Y^i, d) \leq 2 \cdot \frac{1}{d^i} \mod \frac{\mu \cdot d}{d-1}$$

As $i \to \infty$, $2 \cdot \mod \frac{1}{d^{i-1}} \cdot \frac{\mu}{d-1} \to 0$. So, $\lim_{\to i \to \infty}(DS(W, d) - DS(W[i] \cdot Y^i, d)) = 0$. Since $DS(W, d)$ is fixed, $\lim_{\to i \to \infty} DS(W[i] \cdot Y^i, d) = DS(W, d)$.

By definition of convergence, there exists an index $k \geq 0$ such that $DS(W[k] \cdot Y^k, d)$ falls within the $\frac{\mod DS(W,d)}{2}$ neighborhood of $DS(W, d)$. Finally since $DS(W, d) > 0$, $DS(W[k] \cdot Y^k, d) > 0$ as well. But this contradicts our assumption that for all $i \geq 0$, $DS(W[i] \cdot Y^i, d) \leq 0$.

Therefore, DS-comparator with $\leq$ is a safety comparator. ■

Semantically this result implies that for a bounded-weight sequence C and rational discount-factor $d > 1$, if $DS(C, d) > 0$ then C must have a finite prefix C_{pre} such that the discounted-sum of the finite prefix is so large that no infinite extension by bounded weight-sequence Y can reduce the discounted-sum of $C_{\mathsf{pre}} \cdot Y$ with the same discount-factor d to zero or below.

Prior work shows that DS-comparison languages are expressed by Büchi automata iff the discount-factor is an integer [26]. Therefore:

Corollary 6.2 *Let $\mu > 1$ be the upper bound. For* integer *discount-factor $d > 1$*

1. *DS comparators are regular safety for relations $\mathsf{R} \in \{\leq, \geq, =\}$*

2. *DS comparators are regular co-safety for relations $\mathsf{R} \in \{<, >, \neq\}$.*

Proof 39 Immediate from Theorem 6.3, Theorem 4.2 and Corollary 4.2. ■

Lastly, it is worth mentioning that for the same reason [26] DS comparators for non-integer rational discount-factors do not form safety or co-safety automata.

6.4.2 Deterministic DS comparator for integer discount-factor

This section issues deterministic safety/co-safety constructions for DS comparators with integer discount-factors. This is different from prior works since they supply non-deterministic Büchi constructions only [24, 25]. An outcome of DS comparators being regular safety/co-safety (Corollary 6.2) is a proof that DS comparators permit deterministic Büchi constructions, since non-deterministic and deterministic safety automata (and co-safety automata) have equal expressiveness [63]. Therefore, one way to obtain deterministic Büchi construction for DS comparators is to determinize the non-deterministic constructions using standard procedures [63, 80]. However, this will result in exponentially larger deterministic constructions. To this end, this section offers direct deterministic safety/co-safety automata constructions for DS comparator that not only avoid an exponential blow-up but also match their non-deterministic counterparts in number of states (Theorem 6.5).

Key ideas

Due to duality and closure properties of safety/co-safety automata, we only present the construction of deterministic safety automata for DS comparator with upper bound μ, integer discount-factor $d > 1$ and relation $\leq$, denoted by $\mathcal{A}_{\leq}^{\mu,d}$. We proceed by obtaining a *deterministic finite automaton*, (DFA), denoted by $\mathsf{bad}(\mu, d, \leq)$, for the language of bad-prefixes of $\mathcal{A}_{\leq}^{\mu,d}$ (Theorem 6.4). Trivial modifications to $\mathsf{bad}(\mu, d, \leq)$ will furnish the coveted deterministic safety automata for $\mathcal{A}_{\leq}^{\mu,d}$ (Theorem 6.5).

Detailed construction

We begin with some definitions. Let W be a *finite* weight-sequence. By abuse of notation, the discounted-sum of finite-sequence W with discount-factor d is defined as $DS(W, d) = DS(W \cdot 0^{\omega}, d)$. The *recoverable-gap* of a finite weight-sequences W with discount factor d, denoted $\mathsf{gap}(W, d)$, is its normalized discounted-sum: If $W = \varepsilon$ (the empty sequence), $\mathsf{gap}(\varepsilon, d) = 0$, and $\mathsf{gap}(W, d) = d^{|W|-1} \cdot DS(W, d)$ otherwise [31]. Observe that the recoverable-gap has an inductive definition i.e. $\mathsf{gap}(\varepsilon, d) = 0$, where ε is the empty weight-sequence, and $\mathsf{gap}(W \cdot v, d) = d \cdot \mathsf{gap}(W, d) + v$, where $v \in \{-\mu, \ldots, \mu\}$.

This observation influences a sketch for $\mathsf{bad}(\mu, d, \leq)$. Suppose all possible values for recoverable-gap of weight sequences forms the set of states. Then, the transition relation of the DFA can mimic the inductive definition of recoverable gap i.e. there is a transition from state s to t on alphabet $v \in \{-\mu, \ldots, \mu\}$ iff $t = d \cdot s + v$, where s and v are recoverable-gap values of weight-sequences. There is one caveat here: There are infinitely many possibilities for the values of recoverable gap. We need to limit the recoverable gap values to finitely many values of interest. The core aspect of this construction is to identify these values.

First, we obtain a lower bound on recoverable gap for bad-prefixes of $\mathcal{A}^{\mu,d}_{\leq}$:

Lemma 6.4 *Let μ and $d > 1$ be the bound and discount-factor, resp. Let $\mathsf{T} = \frac{\mu}{d-1}$ be the threshold value. Let W be a non-empty, bounded, finite weight-sequence. Weight sequence W is a bad-prefix of $\mathcal{A}^{\mu,d}_{\leq}$ iff $\mathsf{gap}(W,d) > \mathsf{T}$.*

Proof 40 Let a finite weight-sequence W be a bad-prefix of $\mathcal{A}^{\mu,d}_{\leq}$. Then, $DS(W \cdot Y, d) > 0$ for all infinite and bounded weight-sequences Y. Since $DS(W \cdot Y, d) = DS(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d)$, we get $\inf(DS(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d)) > 0 \implies DS(W,d) + + \frac{1}{d^{|W|}} \cdot \inf(DS(Y,d)) > 0$ as W is a fixed sequence. Hence $DS(W,d) + \frac{-\mathsf{T}}{d^{|W|-1}} > 0 \implies \mathsf{gap}(W,d) - T > 0$. Conversely, for all infinite, bounded, weight-sequence Y, $DS(W \cdot Y, d) \cdot d^{|W|-1} = \mathsf{gap}(W,d) + \frac{1}{d} \cdot DS(Y,d)$. Since $\mathsf{gap}(W,d) > T$, $\inf(DS(Y,d)) = -\mathsf{T} \cdot d$, we get $DS(W \cdot Y, d) > 0$. ■

Since all finite and bounded extensions of bad-prefixes are also bad-prefixes, Lemma 6.4 implies that if the recoverable-gap of a finite sequence is strinctly lower that threshold T, then recoverable gap of all of its extensions also exceed T. Since recoverable gap exceeding threshold T is the precise condition for bad-prefixes, all states with recoverable gap exceeding T can be merged into a single state. Note, this state forms an accepting sink in $\mathsf{bad}(\mu, d, \leq)$.

Next, we attempt to merge very low recoverable gap value into a single state. For this purpose, we define *very-good prefixes* for $\mathcal{A}^{\mu,d}_{\leq}$: A finite and bounded weight-sequence W is a *very good* prefix for language of $\mathcal{A}^{\mu,d}_{\leq}$ if for all infinite, bounded extensions of W by Y, $DS(W \cdot Y, d) \leq 0$. A proof similar to Lemma 6.4 proves an upper bound for the recoverable gap of very-good prefixes of $\mathcal{A}^{\mu,d}_{\leq}$:

Lemma 6.5 *Let μ and $d > 1$ be the bound and discount-factor, resp. Let $\mathsf{T} = \frac{\mu}{d-1}$ be*

the threshold value. Let W be a non-empty, bounded, finite weight-sequence. Weight-sequence W is a very-good prefix of $\mathcal{A}^{\mu,d}_{\leq}$ iff $\mathsf{gap}(W,d) \leq -\mathsf{T}$.

Proof 41 Proof is similar to that in Lemma 6.4. ■

Clearly, finite extensions of very-good prefixes are also very-good prefixes. Further, $\mathsf{bad}(\mu,d,\leq)$ must not accept very-good prefixes. Thus, by reasoning as earlier we get that all recoverable gap values that are less than or equal to $-\mathsf{T}$ can be merged into one non-accepting sink state in $\mathsf{bad}(\mu,d,\leq)$.

Finally, for an integer discount-factor the recoverable gap is an integer. Let $\lfloor x \rfloor$ denote the floor of $x \in \mathbb{R}$ e.g. $\lfloor 2.3 \rfloor = 2$, $\lfloor -2 \rfloor = -2$, $\lfloor -2.3 \rfloor = -3$. Then,

Corollary 6.3 *Let μ be the bound and $d > 1$ an integer discount-factor. Let $\mathsf{T} = \frac{\mu}{d-1}$ be the threshold. Let W be a non-empty, bounded, finite weight-sequence.*

- *W is a bad prefix of $\mathcal{A}^{\mu,d}_{\leq}$ iff $\mathsf{gap}(W,d) > \lfloor \mathsf{T} \rfloor$*
- *W is a very-good prefix of $\mathcal{A}^{\mu,d}_{\leq}$ iff $\mathsf{gap}(W,d) \leq \lfloor -\mathsf{T} \rfloor$*

Proof 42 Simply combining the statements in Lemma 6.4 and Lemma 6.5, and using the observation that the gap value is always an integer when the discount factor is an integer. ■

So, the recoverable gap value is either one of $\{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$, or less than or equal to $\lfloor -\mathsf{T} \rfloor$, or greater than $\lfloor \mathsf{T} \rfloor$. This curbs the state-space to $\mathcal{O}(\mu)$-many values of interest, as $\mathsf{T} = \frac{\mu}{d-1} < \frac{\mu \cdot d}{d-1}$ and $1 < \frac{d}{d-1} \leq 2$. Lastly, since $\mathsf{gap}(\varepsilon,d) = 0$, state 0 must be the initial state.

Construction of $\mathsf{bad}(\mu,d,\leq)$ Let μ be the upper bound, and $d > 1$ be the integer discount-factor. Let $\mathsf{T} = \frac{\mu}{d-1}$ be the threshold value. The finite-state automata

$\mathsf{bad}(\mu, d, \leq) = (S, s_I, \Sigma, \delta, \mathcal{F})$ is defined as follows:

- States $S = \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\} \cup \{\mathsf{bad}, \mathsf{veryGood}\}$
- Initial state $s_I = 0$
- Accepting states $\mathcal{F} = \{\mathsf{bad}\}$
- Alphabet $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu - 1, \mu\}$
- Transition function $\delta \subseteq S \times \Sigma \to S$ where $(s, a, t) \in \delta$ then:
  1. If $s \in \{\mathsf{bad}, \mathsf{veryGood}\}$, then $t = s$ for all $a \in \Sigma$
  2. If $s \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$, and $a \in \Sigma$
     (a) If $\lfloor -\mathsf{T} \rfloor < d \cdot s + a \leq \lfloor \mathsf{T} \rfloor$, then $t = d \cdot s + a$
     (b) If $d \cdot s + a > \lfloor \mathsf{T} \rfloor$, then $t = \mathsf{bad}$
     (c) If $d \cdot s + a \leq \lfloor -\mathsf{T} \rfloor$, then $t = \mathsf{veryGood}$

Theorem 6.4 *Let μ be the upper bound, $d > 1$ be the integer discount-factor.* $\mathsf{bad}(\mu, d, \leq)$ *accepts finite, bounded, weight-sequence iff it is a bad-prefix of* $\mathcal{A}^{\mu,d}_{\leq}$.

Proof 43 First note that the transition relation is deterministic and complete. Therefore, every word has a unique run in $\mathsf{bad}(\mu, d, \leq)$. Let last be the last state in the run of finite, bounded weight-sequence W in the DFA. We use induction on the length of W to prove the following:

- $\mathsf{last} \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$ iff $\mathsf{gap}(W, d) = \mathsf{last}$
- $\mathsf{last} = \mathsf{bad}$ iff $\mathsf{gap}(W, d) > \lfloor \mathsf{T} \rfloor$
- $\mathsf{last} = \mathsf{veryGood}$ iff $\mathsf{gap}(W, d) \leq \lfloor -\mathsf{T} \rfloor$

Let $W = \varepsilon$ be a word of length 0. By definition of recoverable gap, $\mathsf{gap}(w, d) = 0$. Also, its run in the DFA $\mathsf{bad}(\mu, d, \leq)$ is the single-state sequence 0, where 0 is the initial state of the DFA.Therefore, the hypothesis is true for the base case.

Let the hypothesis be true for all words of length n. We prove that the hypothesis can be extended to words of length $n + 1$.

Let $W = V \cdot a$ be a word of length $n+1$, where V is a word of length n and last_V be its last state . By induction hypothesis,

- $\mathsf{last}_V \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$ iff $\mathsf{gap}(V, d) = \mathsf{last}_V$
- $\mathsf{last}_V = \mathsf{bad}$ iff $\mathsf{gap}(V, d) > \lfloor \mathsf{T} \rfloor$
- $\mathsf{last}_V = \mathsf{veryGood}$ iff $\mathsf{gap}(V, d) \leq \lfloor -\mathsf{T} \rfloor$

Since, the DFA is deterministic, the last state of W is $\mathsf{last} = \delta(\mathsf{last}_V, a)$.

If $\mathsf{gap}(W, d) > \lfloor \mathsf{T} \rfloor$, we show that $\mathsf{last} = \mathsf{bad}$.

1. If $\mathsf{last}_V = \mathsf{bad}$, then by construction bad is a sink state, in which case $\mathsf{last} = \mathsf{bad}$.
2. If $\mathsf{last}_V = s \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$. By I.H, we know that $s = \mathsf{gap}(V, d)$, therefore by definition of recoverable gap, $\mathsf{gap}(W, d) = d \cdot s + a > \lfloor \mathsf{T} \rfloor$. Therefore, by the transition function rules, $\mathsf{last} = \mathsf{bad}$.
3. If $\mathsf{last}_V = \mathsf{veryGood}$. Since $\mathsf{veryGood}$ is a sink state, none of its transitions will go to state bad.

Conversely, let $\mathsf{last} = \mathsf{bad}$. We show that $\mathsf{gap}(W, d) > \lfloor \mathsf{T} \rfloor$.

1. If $\mathsf{last}_V = \mathsf{bad}$, then by I.H. $\mathsf{gap}(V, d) > \lfloor \mathsf{T} \rfloor$. Since the discount-factor is an integer, V is a bad prefix (Corollary 6.3). Since extensions of bad-prefixes are also bad-prefixes, we get that W is also a bad-prefix. By the same corollary, we get $\mathsf{gap}(W, d) > \lfloor \mathsf{T} \rfloor$.

2. If $\mathsf{last}_V = s \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$. A transition from s goes to bad only if $d \cdot s + a > \lfloor \mathsf{T} \rfloor$. By I.H $s = \mathsf{gap}(V, d)$, therefore $d \cdot s + a = \mathsf{gap}(W, d) > \lfloor \mathsf{T} \rfloor$.

3. If $\mathsf{last}_v = \mathsf{veryGood}$. Not relevant since there are no transition from $\mathsf{veryGood}$ to bad.

The proof for $\mathsf{last} = \mathsf{veryGood}$ iff $\mathsf{gap}(W, d) \leq \lfloor -\mathsf{T} \rfloor$ is similar. It uses the fact that extensions of a very good prefix are also very good prefixes.

The final case is when $\mathsf{last} \in \{\lfloor -\mathsf{T} \rfloor + 1, \ldots, \lfloor \mathsf{T} \rfloor\}$. This is the simplest case since it simply follows the transition relation and inductive definition of recoverable gap.

Sink bad is the only accepting state, a words visits bad iff its recoverable gap is strictly above $\lfloor \mathsf{T} \rfloor$, the DFA accepts the language of bad-prefixes. ■

Finally, the DS comparators are constructed as follows:

Theorem 6.5 *Let μ be the upper bound, and $d > 1$ be the integer discount-factor. DS comparator for all inequalities and equality are either deterministic safety or deterministic co-safety automata with $\mathcal{O}(\mu)$ states.*

Proof 44 This proof should also be treated as an exercise that illustrates that DFA $\mathsf{bad}(\mu, d, \leq)$ is the basis of all our constructions. In fact, the analysis of bad-prefixes and very-good prefixes for the one case of the $\leq$ relation is sufficient.

For $>$: Syntactically, DFA $\mathsf{bad}(\mu, d, \leq)$ is also a deterministic co-safety automaton. We claim that this deterministic co-safety automaton accepts the DS comparison language for the relation $>$. This is true due to the complementation duality of the definitions of safety and co-safety languages. A sequence A is in a co-safety language L iff it has a finite length prefix B that is a bad prefix for the complement safety

language $\overline{L}$. The missing link here is that DS comparison languages for $>$ and DS comparison language for $\leq$ are complements of each other.

For $\leq$: Flipping accepting and non-accepting states of the deterministic co-safety comparator for $>$ will do the trick to generate the deterministic safety comparator for $\leq$ with the same number of states. Another way to think of the construction of the coveted safety comparator is from DFA $\mathsf{bad}(\mu, d, \leq)$ itself. First of all, observe that syntactically the complement of DFA $\mathsf{bad}(\mu, d, \leq)$, denoted by $\overline{\mathsf{bad}(\mu, d, \leq)}$ is also a deterministic safety automata with a single non-accepting sink state. Second of all, semantically DFA $\overline{\mathsf{bad}(\mu, d, \leq)}$ accepts all finite sequences that are not bad-prefixes of the DS comparison language with relation $\leq$. Hence, DFA $\overline{\mathsf{bad}(\mu, d, \leq)}$ is also the deterministic safety automata for DS comparison language for relation $\leq$.

For $<$ and $\geq$: The co-safety and safety automata for $<$ and $\geq$ is obtained by negating the alphabet on all transitions of the DS comparator automata for $>$ and $\leq$, respectively. Alternately, once may construct these comparators from first principles by reasoning over DFA $\mathsf{bad}(\mu, d, \leq)$, as done in the previous two cases.

For $=$ and $\neq$: The default for construction of DS comparators $=$ is to take the intersection of those for $\leq$ and $\geq$. However, that would result in a safety comparator with $\mathcal{O}(\mu^2)$. Similarly, default for $\neq$ would also have $\mathcal{O}(\mu)$ states. Instead, we construct the safety and co-safety comparator for $=$ and $\neq$ in $\mathcal{O}(\mu)$ states as well.

Observe that the bad-prefixes of the DS comparison language for $=$ with either be bad-prefixes of the same for $\leq$ or the bad-prefixes of $\geq$. Recall, that a bounded, finite-length weight sequence W is a bad-prefix of $\leq$ iff $\mathsf{gap}(W, d) > \mathsf{T}$ (Lemma 6.4). Analogously, one can prove that W is a bad-prefix of $\geq$ iff $\mathsf{gap}(W, d) < -\mathsf{T}$. Therefore, if an infinite sequence A has a finite prefix B such that either $\mathsf{gap}(B, d) > \mathsf{T}$ or $\mathsf{gap}(B, d) < -\mathsf{T}$, then the $\neq (DS(A, d) = 0) \equiv DS(A, d) \neq 0$, i.e., is A is present in

the DS comparison language for $\neq$. Therefore, in DFA $\mathsf{bad}(\mu, d, \leq)$, if both bad and $\mathsf{veryGood}$ are accepting, the resulting deterministic Büchi automaton accepts the DS comparison language for $\neq$. Now, this can be converted into a deterministic co-safety automaton, by fusing both the accepting states into one.

Finally, the safety automata for DS comparison language with $=$ simply flips the accepting and non-accepting states of the co-safety automata for $\neq$. ■

As a matter of fact, the most compact non-deterministic DS comparator constructions with parameters μ, d and R also contain $\mathcal{O}(\mu)$ states [24]. In fact, we can go a step further and prove that the deterministic Büchi automaton constructed in Thereom 6.5 are the *minimal* forms, i.e., none of the languages represented by these deterministic automata can be represented by other deterministic automata with fewer number of states. It is worth noting that not all (non-deterministic) Büchi automata have minimal forms. However, deterministic Büchi automata are known to have minimal forms [54, 70].

Theorem 6.6 *The deterministic Büchi automata constructed in Theorem 6.5 for each DS comparator are their respective minimal deterministic Büchi automata.*

Proof 45 Each of the DS comparator constructed in Theorem 6.5 is either a deterministic safety or co-safety automata. Therefore, if these DBAs were not in minimal form, then the underlying deterministic DFA corresponding to the appropriate language of bad-prefixes or their complement would not be in minimal form either.

For instance, if the DBA for DS comparator for $\leq$ or $>$ were not minimal, then the DFA $\mathsf{bad}(\mu, d, \leq)$ would not be a minimal DFA either. We claim that DFA $\mathsf{bad}(\mu, d, \leq)$ is a minimal DFA, and hence it must be the case that the safety/co-safety automata for DS comparison language for $\leq$ or $>$ are minimal DBA as well.

In order to prove that DFA $\mathsf{bad}(\mu, d, \leq)$ is minimal, we prove that no two states are equivalent as per Myhill-Nerode equivalence relation [74], and proving minimal form [59]. As per this relation, two states s and t are said to be equivalent, denoted $s \sim t$, if (a). Either both s and t are accepting states, or both are non-accepting states, and (b). For all alphabet $a \in \Sigma$, if $s \xrightarrow{a} s'$ and $t \xrightarrow{a} t'$ are transitions in the DFA, then $s' \sim t'$ must hold. The core idea behind proving that no two states of $\mathsf{bad}(\mu, d, \leq)$ are equivalent is to realize that the states of $\mathsf{bad}(\mu, d, \leq)$ represent values of the gap value, and transitions are governed by arithmetic on them. So, one can easily find an alphabet $a \in \Sigma$ such that outgoing transitions do not go to equivalent states.

The formal proof can be completed by induction on the minimum distance of states from the single accepting sink state. The inductive hypothesis is that no two states that are at a minimum distance of $k \geq 0$ from the accepting sink state are equivalent. In the base case, k = 0. There is only one such state - the accepting sink state itself- therefore the I.H. holds vacuously. For larger values of k, the intuition described above will demonstrate the veracity of the I.H.

Similar arguments will prove that that the DBAs for all other relations are also minimal. ■

Hence, in this section, not only did we construct safety/co-safety forms for DS comparators, our constructions are also the minimal deterministic representations. These are the most efficient deterministic constructions possible for DS comparators.

6.4.3 QuIPFly: DS inclusion with integer discount factor

Finally, we solve DS inclusion with integer discount factors with using the safety and co-safety comparators presented in the previous Section 6.4.2. Here we will apply the

generic algorithm for quantitative inclusion with regular safety/co-safety comparators (Section 6.3.2) to the case of discounted-sum with integer discount factors. As seen before in Theorem 6.2, the expectation is that the resulting algorithm for DS inclusion with integer discount factor will have an improved complexity. Lastly, we make use of the subset construction based methods in order to present an improved on-the-fly algorithm for DS inclusion, called QuIPFly.

The following statement instantiates Theorem 6.2 for discounted-sum aggregation. Note that the difference in the formal statement is that here we are able to completely characterize the type of the Büchi automaton as well:

Theorem 6.7 *Let P and Q be weighted-automata, and C be a regular safety/co-safety comparator for an inequality with integer discount factor $d > 1$.*

- *Strict DS-inclusion $P \subset Q$ is reduced to emptiness checking of a* safety automaton *of size* $|P||C||Q| \cdot 2^{\mathcal{O}(|Q|\cdot|C|)}$.
- *Non-strict DS-inclusion $P \subseteq Q$ is reduced to emptiness checking of a* weak-Büchi automaton [64] *of size* $|P||C||Q| \cdot 2^{\mathcal{O}(|Q|\cdot|C|)}$.

Proof 46 Unlike Theorem 6.2, the Büchi automaton for emptiness checking has a specific form: safety or weak-Büchi [64]. We can characterize these forms since we know that the DS comparator for strict inequality and equality are co-safety and safety automata respectively (Corollary 6.2).

Recall, that the Büchi automata that is being checked for emptiness is the counterexample automaton. In essence, the counterexample automaton is formed by an intersection of the maximal automaton with an appropriate comparator (Lemma 6.3). Furthermore, we know that when the comparator for non-strict inequality is safety, as for discounted sum, the maximal automaton will be a safety automata (Lemma 6.1).

Consequently, in case of strict DS inclusion, the counterexample automata is constructed by the intersection of a safety maximal automaton with the regular safety DS-comparator for non-strict inequality. Due to closure of safety automata under intersection, the resulting counterexample automata will be a safety automata.

In case of non-strict DS-inclusion, the counterexample automata is constructed by the intersection of the safety maximal automaton with the regular co-safety DS-comparator for strict inequality. As a consequence of Theorem 6.1, the counterexample automaton is weak-Büchi.

Lastly, the size of the resulting safety/weak Büchi automaton is given by the product of the minimal automata (Corollary 6.1) and the comparator. Recall, that the size of all comparators for discounted sum are identical (Theorem 6.5 and Theorem 6.6). Hence, all are given a size of $|C|$. ■

Therefore, the final word is:

Corollary 6.4 *(*[DS-inclusion with safety/co-safety comparator) *Let P, Q be weighted-automata, and C be a regular (co)-safety DS-comparator with integer discount-factor $d > 1$. The complexity of DS-inclusion is $|P||C||Q| \cdot 2^{\mathcal{O}(|Q|\cdot|C|)}$.*

Clearly, this is an improvement over the worst-case complexity of QuIP.

QuIPFly: An on-the-fly implementation

The algorithm for DS inclusion with integer discount factor $d > 1$ proposed in Theorem 6.7 checks for emptiness of the counterexample automata. A naive algorithm will construct the counterexample automata fully, and then check if they are empty by ensuring the absence of an *accepting lasso*.

In QuIPFly, we implement a more efficient algorithm. In our implementation, we

make use of the fact that counterexample automata is a safety/weak Büchi that is constructed from the intersection of a safety maximal automata and an appropriate determinisitc safety/co-safety DS comparator. Recall that maximal automata can be constructed in an on-the-fly fashion since it involves subset construction like techniques. Since, the comparators are also deterministic, they can also be designed in an on-the-fly manner.

In all, this facilitates an *on-the-fly procedure* for DS inclusion, since successor states of state in the counterexample automata can be determined directly from input weighted automata and the comparator automata. The algorithm terminates as soon as an accepting lasso is detected. When an accepting lasso is absent, the algorithm traverses all states and edges of the counterexample automata. Note that for both strict and non-strict DS inclusion, not every lasso in the counterexample automata is an accepting lasso. Only those lassos are accepting for which the loop lies entirely inside one accepting partition of the counterexample automata (recall that states of weak Büchi automata are partitioned, and safety automata are also weak Büchi automata). Tracking whether the loop is in an accepting partition by simply looking at the states in the counterexample automata.

Note that the on-the-fly mechanism does not alter the worst-case performance of DS inclusion, but it contribute to making the algorithm (a) more efficient as it can terminate as soon as an accepting lasso is found, and (b) less memory exhaustive, since it usually doesn't require to store the entire counterexample automata in memory at any given time.

6.5 Empirical evaluation

The goal of the empirical analysis is to examine performance of DS-inclusion with integer discount-factor with safety/co-safety comparators against existing tools to investigate the practical merit of our algorithm. Our optimized on-the-fly algorithm for safety/co-safety comparator-based approach for DS inclusion with integer discount factors is implemented in our prototype called QuIPFly. QuIPFly is written in Python 2.7.12. QuIPFly employs basic implementation-level optimizations to avoid excessive re-computation.

We compare against (a) Regular-comparator based tool QuIP, and (b) DS-determinization and linear-programming tool DetLP. Recall from Chapter 5, QuIP is written in C++, and invokes state-of-the-art Büchi language inclusion-solver RABIT [5]. We enable the `-fast` flag in RABIT, and tune its Java-threads with Xss, Xms, Xmx set to 1GB, 1GB and 8GB, respectively. DetLP is also written in C++, and uses linear programming solver GLPSOL provided by GLPK (GNU Linear Prog. Kit) [2]. We compare these tools along two axes: runtime and number of benchmarks solved.

Figures are best viewed online and in color.

Design and setup for experiments

Due to lack of standardized benchmarks for weighted automata, we follow a standard approach to performance evaluation of automata-theoretic tools [6, 71, 86] by experimenting with *randomly generated* benchmarks, using random benchmark generation procedure described in [24].

The parameters for each experiment are number of states s_P and s_Q of weighted automata, transition density δ, maximum weight wt, integer discount-factor d, and $\mathsf{inc} \in \{\mathsf{strct}, \mathsf{nstrct}\}$. In each experiment, weighted automata P and Q are randomly

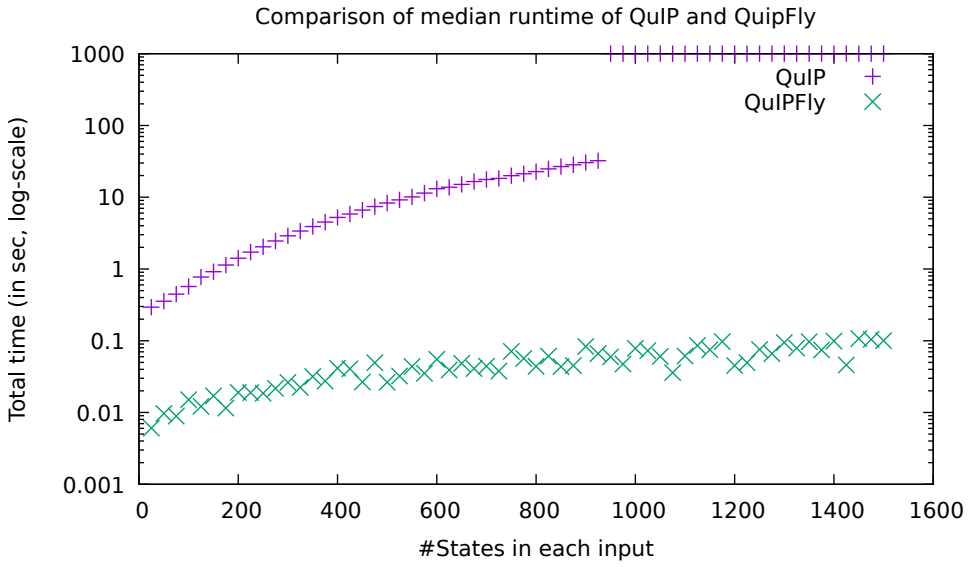


Figure 6.1 : Runtime comparion of QuIP andQuIPFly. $s_P = s_Q$ on x-axis, $wt = 4$, $\delta = 3$, $d = 3$, $P \subset Q$

generated, and runtime of inc-DS-inclusion for all three tools is reported with a time-out of 900sec. We run the experiment for each parameter tuple 50 times. All experiments are run on a single node of a high-performance cluster consisting of two quad-core Intel-Xeon processor running at 2.83GHz, with 8GB of memory per node. We experiment with $s_P = s_Q$ ranging from 0-1500 in increments of 25, $\delta \in \{3, 3.5, 4\}$, $d = 3$, and $wt \in \{d^1 + 1, d^3 - 1, d^4 - 1\}$.

Observations and Inferences

For clarity of exposition, we present the observations for only one parameter-tuple. Trends and observations for other parameters were similar.

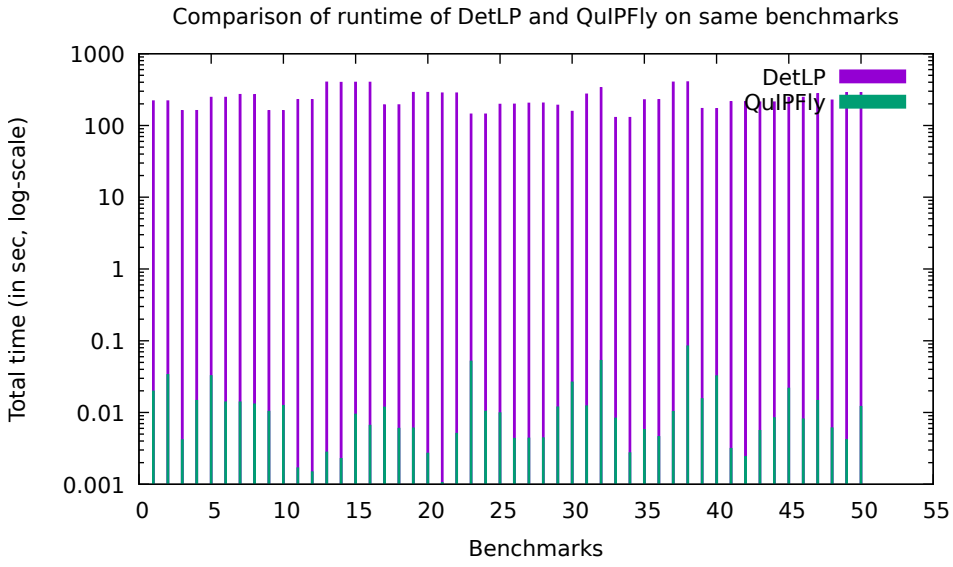


Figure 6.2 : Number of benchmarks solved between DetLP and QuIPFly. $s_P = s_Q = 75$, $wt = 4$, $\delta = 3$, $d = 3$, $P \subset Q$

QuIPFly outperforms QuIP by at least an order of magnitude in runtime. Fig 6.1 plots the median runtime of all 50 experiments for the given parameter-values for QuIP and QuIPFly. More importantly, QuIPFly solves all of our benchmarks within a fraction of the timeout, whereas QuIP struggled to solve at least 50% of the benchmarks with larger inputs (beyond $s_P = s_Q = 1000$). Primary cause of failure is memory overflow inside RABIT. We conclude that regular safety/co-safety comparators outperform their regular counterpart, giving credit to the simpler subset-constructions vs. Büchi complementation.

QuIPFly outperforms DetLP comprehensively in runtime and in number of benchmarks solved. We were unable to plot DetLP in Fig 6.1 since it solved fewer than 50% benchmarks even with small input instances. Fig 6.2 compares the runtime

of both tools on the same set of 50 benchmarks for a representative parameter-tuple on which all 50 benchmarks were solved. The plot shows that QuIPFly beats DetLP by 2-4 orders of magnitude on all benchmarks.

Overall verdict Overall, QuIPFly outperforms QuIP and DetLP by a significant margin along both axes, runtime and number of benchmarks solved. This analysis gives unanimous evidence in favor of our safety/co-safety approach to solving DS-inclusion.

6.6 Chapter summary

The goal of this chapter was to improve comparator-based solutions for DS inclusion. To this end, here we further the understanding of language-theoretic properties of discounted-sum aggregate function by demonstrating that DS-comparison languages form safety and co-safety languages. These properties are utilized to obtain a decision procedure for DS-inclusion that offers both tighter theoretical complexity, improved scalability, and better performance than separation-of-techniques. To the best of our knowledge, this is the first work that applies language-theoretic properties such as safety/co-safety in the context of quantitative reasoning. In doing so, we demonstrate that the close integration of structural analysis and numerical analysis using comparator automata can enhance algorithms in quantitative reasoning since they can benefit from the abundance of work in qualitative reasoning.

Chapter 7

DS inclusion with non-integer discount factors

The theoretical and practical progress in DS inclusion so far have been made for the case when the discount factor is an integer. The problem of complexity of DS inclusion with non-integer discount factors is still very much open - so much so that even its decidability is a mystery so far. In wake of the unknown decidability of DS inclusion with non-integer discount factor, this chapter takes the approach of developing pragmatic solutions to the problem. In this case, we develop an *anytime algorithm* [49] for DS inclusion for non-integer discount factors.

7.1 Introduction

In its most generality, the decidability of DS inclusion is still unknown [39]. The goal of this work is to build a pragmatic solution for DS inclusion despite its unknown decidability. Prior results have shown that the parameter that determines the rate at which weights loose their significance, called the *discount factor*, primarily governs the hardness of DS inclusion: When the discount factor is an integer, DS inclusion is PSPACE-complete [25]; On the contrary, when the discount factor is a non-integer, decidability of DS inclusion is unknown, and has been an open problem for more than a decade [39]. This work strives to address DS inclusion for non-integer discount factors, since most practical applications of DS aggregate function, such as reinforcement learning [85], planning under uncertainty [78], and game theory [75],

make use of non-integer discount factors. In particular, these applications require that the discount factor $d > 1$ to be very close to 1, i.e. $1 < d < 2$. Therefore, this work focuses on DS inclusion when $1 < d < 2$.

This work addresses DS inclusion for non-integer discount factors. In particular, we investigate the case when the discount factor $d > 1$ is very close to 1, i.e. $1 < d < 2$, since that is the value used in practice. Our focus is not on resolving whether DS inclusion is decidable for $1 < d < 2$ but to design *pragmatic solutions* for DS inclusion despite its decidability being unknown.

A natural step in this direction is to solve approximations of DS inclusion. However, we discover that approximations of DS inclusion is at least as hard as DS inclusion itself (Theorem 7.1), since DS inclusion reduces to its approximations in polynomial time. Hence, the decidability of approximations of DS inclusion is also currently unknown. As a result, we are faced with the challenge to design a solution for DS inclusion with $1 < d < 2$ without the ability to develop algorithms for DS inclusion or its approximations.

To address this challenge, we turn to *anytime algorithms* that have been used extensively to design pragmatic solution in AI for problems with high computational complexity or even undecidability [49]. Anytime algorithms are a class of algorithms that generate approximate answers quickly and proceed to construct progressively better approximate solutions over time [49]. In the process, they may either generate an exact solution and terminate, or continuously generate a better approximation. In the context of designing a pragmatic solution for DS inclusion with $1 < d < 2$, we design an anytime algorithm for the same. In addition, our anytime algorithm is *co-computational enumerable*, i.e., it is guaranteed to terminate on input instances on which DS inclusion does not hold. The algorithm gives only approximate answers

otherwise.

The formalism is detailed here. Let P and Q be weighted automata, and let $1 < d < 2$ be the discount factor. We say P *is DS included in* Q, denoted by $P \subseteq Q$, if weight of all executions in P is less than or equal to that in Q. So, our anytime algorithm, called DSInclusion, either terminates and returns a crisp True or False answer to $P \subseteq Q$, or it continuously generates an approximation that P is $d \cdot \varepsilon$-*close* to Q, in a precise sense defined below. If an execution of DSInclusion is interrupted due to external factors such as manual interference or resource overflow, then the algorithm will return the most recently computed $d \cdot \varepsilon$-close approximation. Additionally, if $P \subseteq Q$ does not hold, then the algorithm is guaranteed to terminate after a finite amount of time. Note that we do not prove termination when $P \subseteq Q$ holds. But this is not surprising, because if we could then we would have proven decidability of DS inclusion for $1 < d < 2$.

The core of our algorithm is anytimeInclusion (Overview: § 7.3, Description: § 7.6), a tail recursive algorithm that recurses on the approximation factor $0 < \varepsilon < 1$ by halving it in each new invocation. Conceptually, anytimeInclusion over- and under-approximates DS inclusion in each new invocation of the algorithm, and continues until either an exact solution is obtained or the algorithms' execution is interrupted. Due to the unknown decidability of both approximations, *partial solutions* for the over- and under-approximations are developed. *Comparator automata* [25] based techniques for aggregate functions that represent lower- and upper- approximations of discounted sum are used, respectively for the partial solutions (§ 7.4 and § 7.5, respectively).

Another advantage of our anytime algorithm is that it can be run upto a desired precision $0 < \varepsilon_c < 1$. For this the recursive procedure DSInclusion will be run till

it is invoked with approximation factor ε_c in the worst case, i.e., in the worst-case DSInclusion will be invoked for $\mathcal{O}(\log \frac{1}{\varepsilon_c})$ times, each time with a lower approximation factor. In this case, the complexity of running DSInclusion is linear in $\frac{1}{\varepsilon_c}$, exponential in size of the input DS automata and exponential in $\frac{1}{(d-1)^2}$. This shows that as d and ε_c become smaller solving upto a desired precision explodes rapidly. These observations are expected, as they corroborate with the known result on undecidability of *sum-inclusion* (Loosely speaking, DS inclusion with $d = 1$, and $\varepsilon = 0$) [11, 62].

In conclusion, this work does not resolve the decidability debate on DS inclusion with $1 < d < 2$. Instead, it designs an anytime algorithm that renders approximations with theoretical guarantees, and time-bounds when the precision value is fixed. Thus, our algorithm is suitable for pragmatic purposes.

7.2 Preliminaries

This section defines terminology and notation used in the rest of this chapter. A key difference in this chapter is that we solve DS inclusion over finite words. A similar proof can be adapted for infinite words.

A finite sequence of weights is said to be *bounded* by $\mu > 0$ if the absolute value of all weights in the sequence are less than or equal to μ.

A *finite-state automaton* [87] is a tuple $\mathcal{A} = (S, \Sigma, \delta, \mathit{Init}, \mathcal{F})$, where S is a finite set of *states*, Σ is a finite *input alphabet*, $\delta \subseteq (S \times \Sigma \times S)$ is the *transition relation*, $\mathit{Init} \subseteq S$ is the set of *initial states*, and $\mathcal{F} \subseteq S$ is the set of *accepting states*. A finite-state automaton is *deterministic* if for all states s and inputs a, $|\{s' | (s, a, s') \in \delta\}| \leq 1$ and $|\mathit{Init}| = 1$; otherwise, it is *nondeterministic*. Deterministic and non-determinsitic finite-state automata are denoted by DFA and NFA, respectively. An NFA is *complete* if for all $s \in S$ and $a \in \Sigma$, there exists a transition $(s, a, t) \in \delta$ for some state $t \in S$.

For a word $w = w_0 w_1 \dots w_m \in \Sigma^*$, a *run* ρ of w is a sequence of states $s_0 s_1 \dots s_{m+1}$, such that $s_0 \in \mathit{Init}$, and $\tau_i = (s_i, w_i, s_{i+1}) \in \delta$ for all i. A run ρ is *accepting* if its last state $s_{m+1} \in \mathcal{F}$. A word w is *accepted* by NFA/DFA if it has an accepting run. *Regular languages* are languages accepted by DFA/NFA.

A *weighted automaton over finite words* (weighted automaton, in short), is a tuple $\mathcal{A} = (\mathcal{M}, \gamma, f)$, where $\mathcal{M} = (S, \Sigma, \delta, \mathit{Init}, S)$ is a complete NFA with all states as accepting, $\gamma : \delta \to \mathbb{Z}$ is a *weight function*, and $f : \mathbb{Z}^* \to \mathbb{R}$ is an *aggregation function*. *Words* and *runs* in weighted automata are defined as they are in an NFA. The *weight sequence* of a run $\rho = s_0 s_1 \dots s_{m+1}$ of word $w = w_0 w_1 \dots w_m$ is $wt_\rho = n_0 n_1 \dots n_m$ where $n_i = \gamma(s_i, w_i, s_{i+1})$ for all i. The *weight of a run* ρ, denoted by $f(\rho)$, is $f(wt_\rho)$. The *weight of word* $w \in \Sigma^*$ in weighted automata is defined as $wt(w, \mathcal{A}) = max\{f(\rho) | \rho$ is a run of w in $\mathcal{A}\}$. The problem of f*-inclusion* compares the weight of words in two weighted automata with the same aggregate function. Given weighted automata P and Q with aggregate function $f : \mathbb{N}^* \to \mathbb{R}$, P is said to be f*-included* in Q if for all words $w \in \Sigma^*$, $wt(w, P) \leq wt(w, Q)$.

This work studies the *discounted-sum inclusion* problem. The discounted sum (DS) of a finite sequence $A = a_0 \dots a_n$ for discount factor $d > 1$, denoted $DS(A, d) = a_0 + \frac{a_1}{d} \cdots + \frac{a_n}{d^n}$. *DS automata* with discount factor $d > 1$ are weighted automata with discounted sum aggregation with discount factor d. Therefore, given two DS automata P and Q with the *same* discount factor, P is *DS-included* in Q if weight of every word in P is less than or equal to that in Q. We reserve the notation $P \subseteq Q$ to denote P is DS-included in Q. While DS inclusion is PSPACE-complete when the discount factor is an integer, its decidability is still open for non-integer discount factors [32, 39]. For sake of clearer exposition, this paper assumes that the weight along transitions in a DS automata are non-negative integers. There are no technical

differences in extending the result to integer weights.

In a similar vein to recent progess on DS inlcusion for integer discount factors [24, 28], this work makes use of *comparator automata* to design the anytime algorithm. Comparator automata and comparison languages are defined as follows: For a finite set of integers Σ, an aggregate function $f : \mathbb{Z}^* \to \mathbb{R}$, and equality or inequality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, the *comparison language for f and R* is a language over the alphabet Σ that accepts a word $A \in \Sigma^*$ iff $f(A)\ \mathsf{R}\ 0$ holds. A *comparator automaton* (comparator, in short) for f and R is an automaton that accepts the comparison language for f and R [25]. A comparator is *regular* if its automaton is an NFA. Prior work has shown that comparator for discounted sum (DS comparator, in short) is regular for all R iff the discount factor is an integer [26]. Regularity of the comparator has been crucial to the progress in DS inclusion for integer discount factors. Therefore, another contribution in this work is to define an approximation for discounted-sum with non-integer discount factor so that its comparator is regular. We will use these regular comparators to design the anytime algorithm.

7.3 Overview

Before delving into the technical details of the anytime algorithm, we give a roadmap of the approach. Notions of approximations are central to anytime algorithms, so we begin with defining approximations to DS inclusion:

Definition 7.1 ($d \cdot \varepsilon$-close approximation) Given DS automata P and Q with discount factor $d > 1$, and an approximation parameter $\varepsilon > 0$, P is said to be $d \cdot \varepsilon$-*close to* Q, denoted by $P \subseteq Q + d \cdot \varepsilon$, if for all words $w \in \Sigma^\omega$, $wt(w, P) \leq wt(w, Q) + d \cdot \varepsilon$.

Definition 7.2 ($d \cdot \varepsilon$-far approximation) Given DS automata P and Q with dis-

count factor $d > 1$, and an approximation parameter $\varepsilon > 0$, P is said to be $d \cdot \varepsilon$-*far from* Q, denoted by $P \subseteq Q - d \cdot \varepsilon$, if for all words $w \in \Sigma^\omega$, $wt(w, P) \leq wt(w, Q) - d \cdot \varepsilon$.

Given the open decidability status of DS inclusion, the next alternative is to develop algorithms for the aforementioned approximations. That is not possible either, however, for the following reason:

Theorem 7.1 [Unknown decidability of approximations]

1. *There exists a polynomial time reduction from DS inclusion to $d \cdot \varepsilon$-close approximation for an approximation factor $\varepsilon > 0$.*

2. *There exists a polynomial time reduction from DS inclusion to $d \cdot \varepsilon$-far approximation for an approximation factor $\varepsilon > 0$.*

Proof 47 The intuitive argument behind showing that $P \subseteq Q$ reduces to $P' \subseteq Q' + d \cdot \varepsilon$ for an $\varepsilon > 0$ is to show that one can transform Q into a weighted automaton Q' such that $Q' = Q - d \cdot \varepsilon$. Similarly, to show that $P \subseteq Q$ reduces to $P' \subseteq Q' - d \cdot \varepsilon$, we show that one can transform Q into Q' such that $Q' = Q + d \cdot \varepsilon$. An direct corollary of Theorem 7.1 is that the decidability of approximations of DS inclusion is currently unknown as well.

DS inclusion is decidable iff over approximation of DS is decidable: First we prove that every instance of DS inclusion can be reduced to an instance of over approximation of DS inclusion. Let $d > 1$ be a discount factor and P and Q be DS automata. Then we will show that there exists alternate DS automata R and S, and an approximation factor $\varepsilon > 0$ such that $P \subseteq Q \iff R \subseteq S + d \cdot \varepsilon$. Let $\#$ be an character that is not present in the alphabet of P and Q First of all, we generate a new DS automaton $P_\#$ from P such that every word in P is prefixed with

the character # and its weight is multiplied by $\frac{1}{d}$. This can be done by a simple automata-theoretic transformation of P: Include all states and transitions from P in $P_{\#}$. Retain all accepting states of P in $P_{\#}$. Add a new state $s_{\#}$. Add a transition from $s_{\#}$ to state $s_{v_{init}}$ of P, and assign it a weight of 0. Make the new state $s_{\#}$ the accepting state. This is $P_{\#}$. Similarly, construct DS automaton $Q_{\#}$ from Q. It is easy to see that $P \subseteq Q$ iff $P_{\#} \subseteq Q_{\#}$. Finally, construct Q' from $Q_{\#}$ by assigning the transition from $s_{\#}$ to s_{init} a weight of -1. Then it is easy to see that $P_{\#} \subseteq Q_{\#}$ iff $P_{\#} \subseteq Q' + 1$. Let $\varepsilon = \frac{1}{d}$, then it is easy to see that $P_{\#} \subseteq Q_{\#}$ iff $P_{\#} \subseteq Q' + d \cdot \varepsilon$. Therefore, $P \subseteq Q$ iff $P_{\#} \subseteq Q' + d \cdot \varepsilon$.

Next, we prove that every instance of over-approximation of DS inclusion can be reduced to an instance of DS inclusion. Let P and Q be DS automata with discount factor $d > 1$, and let $\varepsilon > 0$ be its approximation factor. Suppose $P \subseteq Q + d \cdot \varepsilon$ holds. Let $d \cdot \varepsilon = \frac{r}{s}$. Generate $P_{\#}$ and $Q_{\#}$ as earlier. Since $P \subseteq Q + d \cdot \varepsilon$ holds, we get that $P_{\#} \subseteq Q_{\#} + \varepsilon$ holds. Modify $Q_{\#}$ to Q' so that the weight of the transition from its inital state is now ε. Suppose $\varepsilon = \frac{m}{n}$ for natural numbers $m, n > 0$. Then, Multiply the weight of all edges in $P_{\#}$ and Q' with n to obtain new DS automata R and S, respectively. Then, it is easy to see that $P \subseteq Q + d \cdot \varepsilon$ holds then $R \subseteq S$ holds.

DS inclusion is decidable iff under approximation of DS is decidable: Similar to the previous proof except that the "+" will be replace by "-". ■

Given these challenges, we propose an anytime algorithm for DS inclusion. The anytime algorithm either terminates after a finite amount of time with a crisp True or False answer to DS inclusion, or it continuously generates $d \cdot \varepsilon$-close approximations, where the approximation factor $0 < \varepsilon < 1$ decreases in time. If an execution of the algorithm is interrupted at anytime before a natural termination, then it returns the most recently computed $d \cdot \varepsilon$-close approximation.

Algorithm 4 anytimeInclusion(P, Q, d, ε)

Inputs: DS automata P, Q, discount factor $1 < d < 2$, and approximation factor $0 < \varepsilon < 1$

1: **if** lowApproxDSInc(P, Q, d, ε) returns $P \subseteq Q = \mathsf{False}$ **then**
2: **return** $P \subseteq Q = \mathsf{False}$
3: **end if**
4: **if** upperApproxDSInc(P, Q, d, ε) returns $P \subseteq Q = \mathsf{True}$ **then**
5: **return** $P \subseteq Q = \mathsf{True}$
6: **end if**
7: **if** Interrupt **then return** $P \subseteq Q + d \cdot \varepsilon = \mathsf{True}$
8: anytimeInclusion$(P, Q, d, \frac{\varepsilon}{2})$

Algorithm 4 outlines our anytime procedure. On receiving DS automata P and Q with discount factor $1 < d < 2$, the algorithm invokes anytimeInclusion with an initial approximation factor $0 < \varepsilon_{\mathsf{init}} < 1$. As is clear from Algorithm 4, anytimeInclusion is a tail recursive procedure in which the approximation factor is halved in each new invocation. In the invocation with approximation factor $0 < \varepsilon < 1$, anytimeInclusion calls two functions lowApproxDSInc and upperApproxDSInc. Ideally, these procedures would over- and under- approximate DS inclusion using $d \cdot \varepsilon$-close and $d \cdot \varepsilon$-far, respectively. Unfortunately, that is not possible due to Theorem 7.1. Therefore, a challenge here is the design of these two subprocedures. At this point it is sufficient to know that in response to the challenge we design lowApproxDSInc so that it combines partial solutions of DS inclusion and $d \cdot \varepsilon$-close approximation. Specifically, given approximation factor $\varepsilon > 0$, its outcomes are either $P \subseteq Q = \mathsf{False}$ or $P \subseteq Q + d \cdot \varepsilon = \mathsf{True}$. Similarly, the algorithm upperApproxDSInc is designed to return either $P \subseteq Q = \mathsf{True}$ or

$P \subseteq Q - d \cdot \varepsilon = \mathsf{False}$, hence combining DS inclusion and $d \cdot \varepsilon$-far approximation. The algorithm design for lowApproxDSInc and upperApproxDSInc use regular comparator automata for aggregate functions that represent the lower and upper approximations of discounted sum, respectively. These subprocedures have been presented in detail in § 7.4 and § 7.5, respectively.

Equipped with descriptions of lowApproxDSInc and upperApproxDSInc, we can finally describe anytimeInclusion (Algorithm 4). Without loss of generality, suppose that anytimeInclusion can be interrupted only on completion of upperApproxDSInc. Consider the invocation with approximation factor $0 < \varepsilon < 1$. First, lowApproxDSInc will be called. If it returns $P \subseteq Q = \mathsf{False}$, then anytimeInclusion is terminated, and it returns the crisp outcome that $P \subseteq Q = \mathsf{False}$. Otherwise, lowApproxDSInc must have returned $P \subseteq Q + d \cdot \varepsilon = \mathsf{True}$, i.e. the $d \cdot \varepsilon$-close approximation holds. Therefore, if anytimeInclusion is interrupted here onward, it can return this approximate result. But if anytimeInclusion is not interrupted, it proceeds to solve upperApproxDSInc. If upperApproxDSInc returns $P \subseteq Q = \mathsf{True}$, once again anytimeInclusion is terminated, and the crisp $P \subseteq Q = \mathsf{Ture}$ solution is returned. If anytimeInclusion is interrupted at this point, the algorithm returns the $d \cdot \varepsilon$-close approximation result obtained from lowApproxDSInc. Finally, if the algorithm has not been interrupted yet, anytimeInclusion is invoked with lower approximation factor $\frac{\varepsilon}{2}$. That completes the description of our anytime algorithm.

Finally, to see why this algorithm is co-recursively enumerable, observe that if $P \subseteq Q = \mathsf{False}$, then there must be an approximation factor $0 < \gamma < 1$ such that for all $0 < \delta < \gamma$, $P \subseteq Q + d \cdot \delta = \mathsf{False}$. Therefore, as the approximation factor is halved in each invocation of anytimeInclusion, it will eventually be smaller than the aforementioned γ. When this happens, then subprocedure lowApproxDSInc will

be forced to return $P \subseteq Q = \mathsf{False}$. Hence, if $P \subseteq Q = \mathsf{False}$, then the anytime algorithm will necessarily terminate.

7.4 Algorithm lowApproxDSInc

This section describes Algorithm lowApproxDSInc. Recall, given inputs DS automata P and Q, discount factor $1 < d < 2$ and approximation factor $0 < \varepsilon < 1$, $\mathsf{lowApproxDSInc}(P, Q, d, \varepsilon)$ either returns $P \subseteq Q$ does not hold or $P \subseteq Q + d \cdot \varepsilon$ holds. Note that these outcomes are not *mutually exclusive*, i.e., there exist input instances for which both of the outcomes may hold. In these cases, the algorithm may return either of the outcomes; the procedure will still be sound.

Intuitively, lowApproxDSInc solves whether P is f-included in Q, where f is an aggregate function that approximates the discounted-sum from below. Let this aggregate function, denoted DSLow, be defined such that given a weight-sequence W, discount factor $1 < d < 2$ and approximation factor $0 < \varepsilon < 1$, $0 \leq DS(W, d) - \mathsf{DSLow}(W, k, p) < d \cdot \varepsilon$ holds. Then, we argue that if P *is* DSLow-*included in* Q, then $P \subseteq Q + d \cdot \varepsilon$ holds. Otherwise, if P *is not* DSLow-*included in* Q, then $P \subseteq Q$ will not hold. Therefore, lowApproxDSInc solves DSLow-inclusion. We use techniques from regular comparators [25] to solve DSLow-inclusion.

Organization and notation. First, the lower approximation of discounted-sum DSLow is formally defined in § 7.4.1. Second, a regular comparator for DSLow is constructed in § 7.4.2. Finally, we use the regular comparator to design lowApproxDSInc § 7.4.3. Let $k, p > 0$ be positive rationals such that the discount factor $1 < d < 2$ and approximation factor $0 < \varepsilon < 1$ are expressed as $d = 1 + 2^{-k}$ and $\varepsilon = 2^{-p}$, respectively.

7.4.1 Lower approximation of discounted-sum

This section defines lower approximation of discounted sum when $1 < d < 2$. A consideration while defining the aggregate function is that its comparator should be regular. Therefore, our definition of lower approximation of discounted sum is motivated from the notion of *recoverable gap* [31] which is known to play an important role in guaranteeing regularity of comparators [26].

The *recoverable gap* of a weight sequence W w.r.t discount factor $d > 1$ is $d^{|W|-1} \cdot DS(W, d)$. In other words, it is the normalized DS of a weight sequence. Intuitively, the recoverable gap of a weight sequence gives a measure of how far its discounted-sum is from 0, and hence is a building block for designing the comparator automata. A property of recoverable gap that results in the regularity of comparator for DS with integer discount factor is that the minimum non-zero difference between the recoverable gap of sequences is fixed. Specifically, this difference is 1 when the discount factor is an integer. But for non-integer discount factors, this difference can become arbitrarily small [10]. This explains why DS comparator are not regular for non-integer discount factors.

To this effect, we begin by defining an approximation of the recoverable gap such that the aforementioned difference is fixed under the new definition. This is guaranteed by rounding-off the recoverable gap to a fixed *resolution* $r = (d-1) \cdot \varepsilon = 2^{-(p+k)}$, where $d = 1 + 2^{-k}$ is the discount factor and $\varepsilon = 2^{-p}$ is the approximation factor. Formally, let $\mathsf{roundLow}(x, k, p)$ denote the largest integer multiple of the resolution that is less than or equal to x, for $x \in \mathbb{R}$. Then, $\mathsf{roundLow}(x, k, p) = i \cdot 2^{-(p+k)}$ for an integer $i \in \mathbb{Z}$ such that for all integers $j \in \mathbb{Z}$ for which $j \cdot 2^{-(p+k)} \leq x$, we get that $j \leq i$. Then,

Lemma 7.1 *Let $k, p > 0$ be rational-valued parameters. Then, for all real values $x \in \mathbb{R}$, $0 \leq x - \mathsf{roundLow}(x, k, p) < 2^{-(p+k)}$.*

Proof 48 There exists a unique integer $i \in \mathbb{Z}$ and $0 \leq b < 2^{-(p+k)}$ such that $x = i \cdot 2^{-(p+k)} + b$. Then, $\mathsf{roundLow}(x, k, p) = i \cdot 2^{-(p+k)}$. Therefore, we get that $0 \leq x - \mathsf{roundLow}(x, k, p) < 2^{-(p+k)}$. ■

Lemma 7.2 (Monotonicity) *Let $k, p > 0$ be rational-valued parameters. Then, if $x \leq y$, then $\mathsf{roundLow}(x, k, p) \leq \mathsf{roundLow}(y, k, p)$.*

Proof 49 There exist unique integers $i, j, \in \mathbb{Z}$, and positive values $0 \leq a, b < 2^{-(p+k)}$ such that $x = i \cdot 2^{-(p+k)} + a$ and $y = j \cdot 2^{-(p+k)} + b$. By definition of $\mathsf{roundLow}$, $\mathsf{roundLow}(x, k, p) = i \cdot 2^{-(p+k)}$ and $\mathsf{roundLow}(y, k, p) = j \cdot 2^{-(p+k)}$. Then, if $x \leq y$ then one of the two must have occurred:

- $i < j$. In this case, $\mathsf{roundLow}(x, k, p) < \mathsf{roundLow}(y, k, p)$.
- $i = j$ and $a \leq b$. In this case, $\mathsf{roundLow}(x, k, p) = \mathsf{roundLow}(y, k, p)$

Therefore, if $x \leq y$ then $\mathsf{roundLow}(x, k, p) \leq \mathsf{roundLow}(y, k, p)$. ■

Then, for all real values $x \in \mathbb{R}$, $0 \leq x - \mathsf{roundLow}(x, k, p) < 2^{-(p+k)}$. Then, *lower gap* is defined as follows:

Definition 7.3 (Lower gap) Let $k, p > 0$. Let W be a finite weight sequence. The *lower gap* of W with discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$, denoted $\mathsf{gapLower}(W, k, p)$, is

$$\mathsf{gapLower}(W, k, p) = \begin{cases} 0, \text{ for } |W| = 0 \\ \mathsf{roundLow}(\mathsf{gapLower}(U, k, p) + u, k, p) \text{ for } W = U \cdot u \end{cases}$$

Note that the minimum non-zero difference between the lower gap of weight sequences is the resolution $r = 2^{-(p+k)}$. Similar to the relationship between recoverable gap and DS, lower approximation of DS is defined as follows:

Definition 7.4 (Lower approximation of discounted-sum) Let $k, p > 0$. Let W be a finite weight sequence. The *lower approximation of discounted sum*, called lower DS, for weight sequence W with discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ is denoted by and defined as

$$\mathsf{DSLow}(W, k, p) = \mathsf{gapLower}(W, k, p)/d^{|W|-1}$$

To complete the definition, we prove that the value computed by DSLow in Definition 7.4 corresponds to a value close to the discounted-sum. To prove that we first establish the following" Given discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$, a *resolution sequence of length* $n > 0$, denoted R_n, is the n-length sequence in which all elements are $r = 2^{-(p+k)}$.

Lemma 7.3 *Let $k, p > 0$ be rational-valued parameters. Let $d = 1 + 2^{-k}$ be the non-integer, rational discount factor and $\varepsilon = 2^{-p}$ be the approximation factor. Let W be a finite non-empty weight sequence. Then $0 \leq \mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) < \mathsf{gap}(R_{|W|}, d)$.*

Proof 50 We prove the above by induction on the length sequence W.

Base case: When $|W| = 1$. Let $W = w_0$. In this case, $\mathsf{gap}(W, d) = w_0$ and $\mathsf{gapLower}(W, k, p) = \mathsf{roundLow}(w_0, k, p)$. Then, from Lemma 7.1 we get that $0 \leq \mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) < r$, which in turn is the same as $0 \leq \mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) < \mathsf{gap}(R_1, d)d$.

Inductive hypothesis: For all weight-sequences W of length $n \geq 1$, it is true that $0 \leq \mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) < \mathsf{gap}(R_n, d)$.

Induction step: We extend this result to weight-sequences of length $n+1$. Let W be an $n+1$-length weight-sequence. Then $W = W[n] \cdot w_n$, where $W[n]$ is the n-length prefix of W and w_n is $n+1$-th element.

We first show that $\mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) \geq 0$:

$$\begin{aligned}
&\mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) \\
=&d \cdot \mathsf{gap}(W[n], d) + w_n - \mathsf{roundLow}(d \cdot \mathsf{gapLower}(W[n], k, p) + w_n, k, p)
\end{aligned}$$

Using monotonicity of $\mathsf{roundLow}$ and the inductive hypothesis, we get

$$\geq d \cdot \mathsf{gap}(W[n], d) + w_n - \mathsf{roundLow}(d \cdot \mathsf{gap}(W[n], d) + w_n, k, p)$$

From Lemma 7.1, we get the desired result.

Next, we show that $\mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) < \mathsf{gap}(R_{n+1}, d)$.

$$\begin{aligned}
&\mathsf{gap}(W, d) - \mathsf{gapLower}(W, k, p) \\
=&d \cdot \mathsf{gap}(W[n], d) + w_n - \mathsf{roundLow}(d \cdot \mathsf{gapLower}(W[n], k, p) + w_n, k, p)
\end{aligned}$$

From Lemma 7.1, we get

$$\begin{aligned}
<&d \cdot \mathsf{gap}(W[n], d) + w_n - (d \cdot \mathsf{gapLower}(W[n], k, p) + w_n) + 2^{-(p+k)} \\
=&d \cdot \mathsf{gap}(W[n], d) - d \cdot \mathsf{gapLower}(W[n], k, p) + 2^{-(p+k)}
\end{aligned}$$

From the inductive hypothesis, we get

$$\begin{aligned}
<&d \cdot \mathsf{gap}(R_n, d) + 2^{-(p+k)} \text{ where } R_n \text{ is the } n\text{-length resolution sequence} \\
=&\mathsf{gap}(R_{n+1}, d) \text{ where } R \text{ is the } n+1\text{-length resolution sequence}
\end{aligned}$$

This completes both sides of the proof. ■

Theorem 7.2 *Let $d = 1 + 2^{-k}$ be the discount factor and $\varepsilon = 2^{-p}$ be the approximation factor, for rationals $p, k > 0$. Then for all weight sequences W, $0 \leq DS(W,d) - \mathsf{DSLow}(W,k,p) < d \cdot \varepsilon$.*

Proof 51 The statement clearly holds when $|W| = 0$ since $DS(W,d) = \mathsf{DSLow}(W,k,p) = 0$. For $|W| > n$, we have proven that $0 \leq \mathsf{gap}(W,d) - \mathsf{gapLower}(W,k,p) < \mathsf{gap}(R_{|W|},d)$, where $R_{|W|}$ is the $|W|$-length sequence in which all elements are equal to $r = 2^{-(p+k)}$ (Lemma 7.3). Finally, division by $d^{|W|-1}$ completes the proof.

The complete details are as follows: When $|W| = 0$, $DS(W,d) = \mathsf{DSLow}(W,k,p) = 0$, since $\mathsf{gap}(W,d) = \mathsf{gapLower}(W,k,p) = 0$. Therefore, $0 \leq DS(W,d) - \mathsf{DSLow}(W,k,p) < d \cdot 2^{-p}$ holds when $|W| = 0$.

Otherwise, from Lemma 7.3, we get that $0 \leq \mathsf{gap}(W,d) - \mathsf{gapLower}(W,k,p) < \mathsf{gap}(R_n,d)$, where $n = |W|$. On division by d^{n-1}, we get that $0 \leq DS(W,d) - \mathsf{DSLow}(W,k,p) < DS(R,d)$. Now $DS(R,d) \leq DS(R_\infty,d)$, where R_∞ is the ∞-length resolution sequence. Now, $DS(R_\infty,d) = \frac{2^{-(p+k)} \cdot d}{d-1} = \frac{(d-1) \cdot \varepsilon \cdot d}{d-1} < \varepsilon \cdot d$. Therefore, we get the desired result that $0 \leq DS(W,d) - \mathsf{DSLow}(W,k,p) < d \cdot \varepsilon$.

■

Therefore, the lower approximation of DS is well defined in this section.

7.4.2 Comparator for lower approximation of DS

This section covers the construction of the comparator automaton for the lower approximation of discounted-sum from Defintion 7.4. We show that the comparator is regular by explicitly constructing its DFA. The construction will utilize the fixed non-zero minimum property of the lower gap.

We begin with formal definitions of the comparison language and comparator automata for lower DS.

Definition 7.5 (Comparison language for lower approximation of DS) Let $\mu > 0$ be an integer bound, and k, p be positive integers. The *comparison language for lower approximation of discounted sum* with discount factor $d = 1 + 2^{-k}$, approximation factor $\varepsilon = 2^{-p}$, upper bound μ and inequality relation $\mathsf{R} \in \{\leq, \geq\}$ is a language of finite weight sequences W over the alphabet $\Sigma = \{-\mu, \ldots, \mu\}$ that accepts W iff $\mathsf{DSLow}(W, k, p)$ R 0 holds.

Definition 7.6 (Comparator automata for lower approximation of DS) Let $\mu > 0$ be an integer bound, and k, p be positive integers. The *comparator automata for lower approximation of discounted sum* with discount factor $d = 1 + 2^{-k}$, approximation factor $\varepsilon = 2^{-p}$, upper bound μ and inequality relation $\mathsf{R} \in \{\leq, \geq\}$ is an automaton that accepts the corresponding comparison language.

Next, we construct a DFA for the comparator for lower DS.

The first observation towards the construction is that $\mathsf{DSLow}(W, k, p)$ R 0 iff $\mathsf{gapLower}(W, k, p)$ R 0, for all finite weight sequences W. Therefore, it is sufficient to construct a DFA that accepts weight sequence W iff $\mathsf{gapLower}(W, k, p)$ R 0. We achieve this by (a). Creating one state of the DFA for every possible value of lower gap. (b). Note that the definition of lower gap (Definition 7.3) is inductive on the length of the weight sequence. So, transitions between states is defined so that they obey the inductive definition. For (a), note that lower gap are of the form $i \cdot r$ where $i \in \mathbb{Z}$. Therefore, for all $i \in \mathbb{Z}$, we introduce a state i to represent the lower gap $i \cdot r$. For (b). we include a transition from state i to state j on symbol $a \in \Sigma = \{-\mu, \ldots, \mu\}$ iff $j \cdot r = \mathsf{gapLower}(i \cdot r + a, k, p)$, thereby following Definition 7.3. Note that this

equation makes the transition relation deterministic as for all $i \in \mathbb{Z}$ and $a \in \Sigma$ there is a unique $j \in \mathbb{Z}$ that satisfies it. Since $\mathsf{gapLower}(W, k, p) = 0$ when $|W| = 0$, state 0 is made the initial state. Finally, a state i is an accepting state iff i R 0 holds.

The automaton created above has infinitely many states as every possible value of lower gap corresponds to a state. To obtain finitely many states, we show that it is sufficient to consider finitely many values of the lower gap:

Lemma 7.4 (Bounds on lower gap-value) *Let $\mu > 0$ be an integer bound. Let k, p be positive integers s.t. $d = 1 + 2^{-k}$ is the discount factor, and $\varepsilon = 2^{-p}$ is the approximation factor. Let W be a finite and bounded weight sequence.*

1. *If $\mathsf{gapLower}(W, k, p) \leq -\mu \cdot 2^k$ then for all $u \in \{-\mu, \ldots, \mu\}$, $\mathsf{gapLower}(W \cdot u, k, p) \leq -\mu \cdot 2^k$.*

2. *If $\mathsf{gapLower}(W, k, p) \geq \mu \cdot 2^k + 2^{-p}$ then for all $u \in \{-\mu, \ldots, \mu\}$, $\mathsf{gapLower}(W \cdot u, k, p) \geq \mu \cdot 2^k + 2^{-p}$.*

Proof 52 Proof for (1.). Recall, $\mathsf{gapLower}(W \cdot u, k, p) = \mathsf{roundLow}(d \cdot \mathsf{gapLower}(W, k, p) + u, k, p)$. From the definition of $\mathsf{roundLow}$, we get that $\mathsf{gapLower}(W \cdot u, k, p) \leq d \cdot \mathsf{gapLower}(W, k, p) + u$. Since $\mathsf{gapLower}(W, k, p) \leq -\mu \cdot 2^k$ holds, we get $\mathsf{gapLower}(W \cdot u, k, p) \leq d \cdot (-\mu \cdot 2^k) + u = (1 + 2^{-k}) \cdot (-\mu \cdot 2^k) + u = -\mu \cdot 2^k - \mu + u$. Since u is at most μ, we get that $\mathsf{gapLower}(W \cdot u, k, p) \leq -\mu \cdot 2^k$.

Proof of (2.) follows similarly, and hence has been omitted. ■

The outcome of Lemma 7.4 is that it is sufficient to track the lower gap value only when it is between $-\mu \cdot 2^k$ and $\mu \cdot 2^k + 2^{-p}$ in the construction.

Construction

Let $\mu > 0$, $d = 1 + 2^{-k}$, $\varepsilon = 2^{-p}$ be the upper bound, discount factor and approximation factor, respectively. Let T_l be the largest integer such that $\mathsf{T}_l \cdot 2^{-(p+k)} \leq -\mu \cdot 2^k$ (Lemma 7.4- Part 1). Let T_u be the smallest integer such that $\mathsf{T}_u \cdot 2^{-(p+k)} \geq \mu \cdot 2^k + 2^{-p}$ (Lemma 7.4 - Part 2). For relation $\mathsf{R} \in \{\leq, \geq\}$, construct DFA $\mathsf{compLow}(\mu, k, p, \mathsf{R}) = (S, s_I, \Sigma, \delta, \mathcal{F})$ as follows:

- $S = \{\mathsf{T}_l, \mathsf{T}_l + 1, \ldots, \mathsf{T}_u\}$, $s_I = \{0\}$ and $\mathcal{F} = \{i | i \in S \text{ and } i \ \mathsf{R}\ 0\}$
- Alphabet $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu - 1, \mu\}$
- Transition function $\delta \subseteq S \times \Sigma \times S$ where $(s, a, t) \in \delta$ then:
  1. If $s = \mathsf{T}_l$ or $s = \mathsf{T}_u$, then $t = s$ for all $a \in \Sigma$
  2. Else, let $\mathsf{roundLow}(d \cdot s \cdot 2^{-(p+k)} + a, k, p) = i \cdot 2^{-(p+k)}$ for $i \in \mathbb{Z}$
     (a) If $\mathsf{T}_l \leq i \leq \mathsf{T}_u$, then $t = i$
     (b) If $i > \mathsf{T}_u$, then $t = \mathsf{T}_u$
     (c) If $i < \mathsf{T}_l$, then $t = \mathsf{T}_l$

Theorem 7.3 *Let $\mu > 0$ be and integer upper bound. Let $k, p > 0$ be rational parameters s.t. $d = 1 + 2^{-k}$ is the discount factor and $\varepsilon = 2^{-p}$ is the approximation parameter. DFA $\mathsf{compLow}(\mu, k, p, \mathsf{R})$ accepts a finite weight sequence $W \in \Sigma^*$ iff $\mathsf{DSLow}(W, k, p)\ \mathsf{R}\ 0$. DFA $\mathsf{compLow}(\mu, k, p, \mathsf{R})$ has $\mathcal{O}(\mu \cdot 2^{2k+p})$ states.*

Proof 53 The proof shows that the final state of a the run of a word represents its lower gap value. For this we show three things: Let s_f be the final state of the run. (a). if $\mathsf{T}_l < s_f < \mathsf{T}_u$ then its lower gap value is $s_f \cdot r$, (b). if $\mathsf{T}_l \geq s_f$ then the lower

gap value is less than or equal to $\mathsf{T}_l \cdot r$, and (c). if $\mathsf{T}_u \leq s_f$ then the lower gap value is greater than or equal to $\mathsf{T}_u \cdot r$.

The proof is very similar to Theorem 6.4, and hence its details have been left for the reader to fill in. ■

7.4.3 Algorithm details for lowApproxDSInc

This section describes lowApproxDSInc. Recall, given DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor 2^{-p}, lowApproxDSInc returns either $P \subseteq Q = \mathsf{False}$ or $P \subseteq Q + d \cdot \varepsilon = \mathsf{True}$. In cases where both outcomes are possible, lowApproxDSInc may return either of the outcomes. In our design of lowApproxDSInc, it performs DSLow-inclusion between the DS automaton, as justified in Lemma 7.5-7.6. Subsequently, the algorithm uses the regular comparator for DSLow to design its inclusion procedure (Algorithm 5). Lastly, we illustrate a property of lowApproxDSInc that is necessary in our anytime procedure for DS inclusion to become co-recursively enumerable (Theorem 7.6).

Terminology: A run ρ_P of word w in P is said to be *dominated* by Q if there exists a run ρ_Q in Q on the same word such that $\mathsf{DSLow}(\rho_P - \rho_Q, k, p) \leq 0$.

Lemma 7.5 *Given DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$.*

1. *If all runs in P are dominated by Q, then $P \subseteq Q + d \cdot \varepsilon$ holds.*
2. *If there exists a run in P that is not dominated by Q, then $P \subseteq Q$ does not hold.*

Proof 54 Proof of (1.): Let for all words $w \in \Sigma^*$, for all runs of w $\rho_P \in P$, there exists a run of w $\rho_Q \in Q$ such that $\mathsf{DSLow}(\rho_P - \rho_Q, k, p) \leq 0$ be true. Then $\mathsf{DSLow}(\rho_P -$

$\rho_Q, k, p) \leq 0$ implies that $DS(\rho_P - \rho_Q, d) \leq d \cdot \varepsilon \equiv DS(\rho_P, d) \leq DS(\rho_Q, d) + d \cdot \varepsilon$. Since weight of a word is given by the maximum weight of its all runs, we get that for all word $w \in \Sigma^*$, $wt_P(w) < wt_Q(w) + d \cdot \varepsilon$. Therefore, $P \subseteq Q + d \cdot \varepsilon$ holds.

Proof of (2.): Let $w \in \Sigma^*$ be the word for which there exists a run of w $\rho_P \in P$ such that for all runs of $w\rho_Q \in Q$, $\mathsf{DSLow}(\rho_P - \rho_Q, k, p) > 0$ holds. $\mathsf{DSLow}(\rho_P - \rho_Q, k, p) > 0$ implies $DS(\rho_P - \rho_Q, d) > 0 \equiv DS(\rho_P, d) > DS(\rho_Q, d)$. Since weight of a word is given by the maximum weight of its all runs, we get that there exists word $w \in \Sigma^*$, $wt_P(w) > wt_Q(w)$. So, $P \subseteq Q$ does not hold. ■

Therefore, $\mathsf{lowApproxDSInc}(P, Q, d, \varepsilon)$ is designed such that it returns True iff all runs in P are dominated by Q. To attain this, we construct NFA $\mathsf{dominated}$ in Algorithm 5 so that it contains all runs in P that are dominated by Q. This construction utilizes the comparator automata for lower DS. Lastly, language inclusion between $\mathsf{dominated}$ and NFA $\hat{P}_{-wt}$, that consists of all runs of P, determines if all runs of P are dominated or not. Recall, currently we assume all weights in the DS automata are non-negative integers. As a result, the upper bound for comparator construction is set to μ, where $\mu > 0$ is the maximum weight along all transitions in both DS automata. In case the weights along transitions are non-negative, then the comparator will be constructed with upper bound $2 \cdot \mu > 0$, where μ is the maximum of absolute values of weight along all transitions in both DS automata. The rest will be identical.

Algorithm Details

For DS automaton P, procedure $\mathsf{AugmentWtAndLabel}(P)$ generates an NFA $\hat{P}$ by converting transition $s \xrightarrow{a} t$ with weight wt and unique transitions label l in P to a transition $s \xrightarrow{a,wt,l} t$ in $\hat{P}$ (Line 1). Procedure $\mathsf{productDif}(\hat{P}, Q)$ generates the product of NDA $\hat{P}$ with DS automaton Q in an NDA $\hat{P} - Q$ such that it also records

the difference of weight of transitions. Therefore, if $s \xrightarrow{a,wt_1,l} t$ and $p \xrightarrow{a} q$ with weight wt_2 are transitions in $\hat{P}$ and Q, respectively, then $(s,p) \xrightarrow{a,wt_1-wt_2,l} (t,q)$ is a transition in $\hat{P}-Q$. All states in NFA $\hat{P}-Q$ are accepting states. From § 7.4.1 we know that approxDSComp is the comparator for lower approximation of DS with upper bound μ, discount factor d, approximation factor ε and relation $\leq$. NDA dominatedWitness forms the intersection of $\hat{P}-Q$ with the comparator by matching the weight-component in $\hat{P}-Q$ with the weight-alphabet in the comparator. Therefore, if $s \xrightarrow{a,wt,l} t$ and $p \xrightarrow{wt} q$ are transitions in $\hat{P}-Q$ and the comparator, respectively, then $(s,p) \xrightarrow{a,wt,l} (t,q)$ is a transition in the intersection. Furthermore, a state (r,s) is accepting iff both r and s are accepting states in their respective NFA. Finally, dominated and $\hat{P}_{-wt}$ are obtained by projecting out the weight component from dominatedWitness and $\hat{P}$, respectively. Specifically, if $s \xrightarrow{a,wt,l} t$ is a transition in the original automaton, then $s \xrightarrow{a,l} t$ is a transition in the final automata. Here, $\hat{P}_{wt} \subseteq$ dominated refers to language inclusion between the two NFAs.

Lemma 7.6 *Given DS automata P and Q, discount factor $d = 1+2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$,* lowApproxDSInc(P, Q, d, ε) *returns* True *iff all runs in P are dominated by Q.*

Proof 55 From the algorithm, it is clear that there is a one-one correspondence between words $(w, L) \in \hat{P}_{-wt}$ and runs ρ_P of word w in P for all words $w \in \Sigma^*$.

From the algorithm, it is also clear that $(w, L) \in$ dominated iff $w \in \Sigma^*$, with a run $\rho_P \in P$ which has been labelled by L such that there exists a run $\rho_Q \in Q$ of w such that DSLow$(\rho_P - \rho_Q, k, p) \leq 0$.

Then aux == True iff $\hat{P}_{-wt} \subseteq$ dominated. By definition of language inclusion, this holds iff for all $(w, L) \in \hat{P}_{-wt}$ we get that $(w, L) \in$ dominated. From the one-one

Algorithm 5 lowApproxDSInc(P, Q, d, ε)

Inputs: DS automata P, Q, discount factor $1 < d < 2$, approximation factor $0 < \varepsilon < 1$

1: $\hat{P} \leftarrow$ AugmentWtAndLabel(P)

2: $\hat{P} - Q \leftarrow$ productDif$(\hat{P}, Q)$

3: dominatedWitness $\leftarrow$ Intersect$(\hat{P} - Q,$ compLow$(\mu, \log \frac{1}{d-1}, \log \frac{1}{\varepsilon}, \leq))$, where μ is the maximum of the absolute value of weights in $\hat{P} - Q$

4: dominated $\leftarrow$ Project(dominatedWitness)

5: aux $\leftarrow \hat{P}_{-wt} \subseteq_{\mathsf{LI}}$ dominated // $\subseteq_{\mathsf{LI}}$ refers to Language inclusion

6: **if** aux **then**

7: **return** $P \subseteq Q + d \cdot \varepsilon =$ True

8: **else**

9: **return** $P \subseteq Q =$ False

10: **end if**

correspondence between words in $\hat{P}_{-wt}$ and runs in P, we get that for all $w \in \Sigma^*$, for all runs $\rho_P \in P$ of w, let (w, L) be its correspondence in $\hat{P}_{-wt}$ then $(w, L) \in$ dominated. By the condition under which a word is a member of dominated we get that $w \in \Sigma^*$, for all runs $\rho_P \in P$ of w, such that there exists a run $\rho_Q \in Q$ of w such that DSLow$(\rho_P - \rho_Q, k, p) \leq 0$. ■

Theorem 7.4 [Soundness] *For all inputs DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$, algorithm* lowApproxDSInc *is sound.*

Proof 56 This follows directly from Lemma 7.5 and Lemma 7.6. ■

Theorem 7.5 [Complexity] *Given DS automata P and Q, discount factor $d = 1 +$*

2^{-k} *and approximation factor* $\varepsilon = 2^{-p}$ *for rational values* $k, p > 0$. *Let* μ *be the absolute value of the largest weight in* P *and* Q. *Then the worst case complexity of* lowApproxDSInc *is* $2^{\mathcal{O}(n)}$ *where* $n = |P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon}$.

Proof 57 The size of dominated is $\mathcal{O}(|P| \cdot |Q| \cdot \mathsf{compLow}(\mu, \log(\frac{1}{\mathsf{d}-1}), \log(\frac{1}{\varepsilon})))$, which is equal to $\mathcal{O}(|P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon})$. Then, the complexity of $\hat{P}_{-wt} \subseteq$ dominated is $|P| \cdot 2^{\mathcal{O}(|P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon})}$. This equates to $2^{\mathcal{O}(|P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon} + \log(|P|)}$. Keeping the dominating terms in the exponent, we get the worst-case complexity to be $2^{\mathcal{O}(|P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon})}$. ■

Lastly, we show that if $P \subseteq Q$ does not hold, there exists a sufficiently small approximation factor $\varepsilon > 0$ such that when lowApproxDSInc is invoked with ε, it returns that $P \subseteq Q$ does not hold. This property will be crucial in proving co-recursive enumerability of our anytime procedure for DS inlcusion.

Theorem 7.6 *(*Bias*) Given DS automata* P, Q, *and discount factor* $1 < d < 2$. *If* $P \subseteq Q =$ False, *there exists an approximation factor* $0 < \varepsilon < 1$ *such that for all* $0 < \gamma < \varepsilon$, lowApproxDSInc(P, Q, d, γ) *returns* $P \subseteq Q =$ False.

Proof 58 The core idea is that when $P \subseteq Q$ does not hold, then there must exist a word $w \in \Sigma^*$ such that $wt(w, P) > wt(w, Q) + d \cdot \delta$. Therefore, for a sufficiently low value of ε, $P \subseteq Q + d \cdot \varepsilon =$ False. Then, for these values of ε, lowApproxDSInc will necessarily return $P \subseteq Q =$ False.

Since $P \subseteq Q =$ False there exists a word $w \in \Sigma^*$ such that $wt(w, P) = wt(w, Q) + d \cdot \gamma$ for a rational value $\gamma > 0$. Since weight of words is computed as the maximum of weight of its runs, there must exist a run ρ_P of w in P such that for all runs ρ_Q of w in Q, we get that such that $DS(\rho_P, d) - DS(\rho_Q, d) > d \cdot \gamma = DS(\rho_P - \rho_Q, d) > d \cdot \gamma$. Let $k, p > 0$ be rational values such that $d = 1 + 2^{-k}$ and $\gamma = 2^{-p}$. From Theorem 7.2,

we get that for all $q \geq p+1$ $\mathsf{DSLow}(\rho_P - \rho_Q, k, q) > d \cdot \frac{\gamma}{2}$. Therefore, from Lemma 7.5 we get that for all $q \geq p+1$, $\mathsf{lowApproxDSInc}(P, Q, d, q) = \mathsf{False}$. ■

7.5 Algorithm upperApproxDSInc

This section describes Algorithm upperApproxDSInc - the second sub-procedures in our anytime algorithm for DS inclusion. Given inputs DS automata P and Q, discount factor $1 < d < 2$ and approximation factor $0 < \varepsilon < 1$, upperApproxDSInc (P, Q, d, ε) either returns $P \subseteq Q$ holds or $P \subseteq Q - d \cdot \varepsilon$ does not hold. As earlier, these outcomes are not *mutually exclusive*. In these cases, the algorithm may return either of the outcomes as they are both sound.

The design of upperApproxDSInc follows that of lowApproxDSInc very closely. Intuitively, upperApproxDSInc solves whether P is f-included in Q where aggregate function f is the upper approximation of discounted-sum. Notice how similar this is to the intuition behind lowApproxDSInc. As a result, upperApproxDSInc follows the same three stages as earlier: (a). Define the upper approximation of discounted-sum, (b). Construct its regular comparator, and (c). Use the regular comparator to design upperApproxDSInc. Each of these individual steps are very similar to those in the previous section. As a result, the critical distinctions are highlighted first, and then the details are given. One may skip the details to avoid repetition. The details are mentioned here for sake of completeness.

The first distinction is in the definition of the upper approximation of discounted. It is similar to that of the lower approximation for DS except that it makes use of an *upper gap*. In turn, the upper gap of a value is defined similar to the lower gap except that the upper gap is rounded-off to the smallest multiple of $2^{-(p+k)}$ that is greater than or equal to the value, where p and k are defined as earlier. Using a

similar vein of reasoning as in § 7.4.2, the comparator for the upper approximation can also be constructed. Second, algorithm upperApproxDSInc is almost identical to lowApproxDSInc in Algorithm 5 except that in Line 3 algorithm upperApproxDSInc constructs the regular comparator for the upper approximation of discounted sum.

Yet, another important distinction between lowApproxDSInc and upperApproxDSInc is that if $P \subseteq Q = \mathsf{True}$ holds, then upperApproxDSInc cannot guarantee that for a small enough value of the approximation factor upperApproxDSInc with return $P \subseteq Q = \mathsf{True}$. The core idea here is that if $P \subseteq Q = \mathsf{True}$ then the difference between words in P and in Q could be arbitrarily small. In particular, for every possible value of the approximation factor, there may be a word for which the difference in its weight in P and Q is smaller than the approximation factor. As a result, the option of $P \subseteq Q - d \cdot \varepsilon = \mathsf{False}$ may get triggered, never returning the outcome that $P \subseteq Q = \mathsf{True}$ holds.

The rest of this section gives all details of upperApproxDSInc.

7.5.1 Upper approximation of discounted-sum

In the first stage we define the upper approximation of discounted-sum so that its recoverable gap obeys the bounded non-zero minimal difference property.

For a rational number $x \in \mathbb{Q}$, let $\mathsf{roundUpper}(x, k, p)$ denote the smallest integer multiple of resolution that is more than or equal to x. Formally, $\mathsf{roundUpper}(x, k, p) = i \cdot 2^{-(p+k)}$ for an integer $i \in \mathbb{Z}$ such that for all $j \in \mathbb{Z}$, $j \cdot 2^{-(p+k)} \geq x$ implies $i \leq j$. The upper gap value and upper approximation of discounted sum are defined as follows:

Lemma 7.7 *Let $k, p > 0$ be rational-valued parameters. Then, for all real values $x \in \mathbb{R}$, $0 \leq \mathsf{roundUpper}(x, k, p) - x < 2^{-(p+k)}$.*

Proof 59 There exists a unique integer $i \in \mathbb{Z}$ and $0 \leq b < 2^{-(p+k)}$ such that $x = i \cdot 2^{-(p+k)} - b$. Then, $\mathsf{roundUpper}(x,k,p) = i \cdot 2^{-(p+k)}$. Therefore, we get that $0 \leq \mathsf{roundUpper}(x,k,p) - x < 2^{-(p+k)}$. ■

Lemma 7.8 (Monotonicity) *Let $k, p > 0$ be rational-valued parameters. Then, if $x \geq y$, then $\mathsf{roundLow}(x,k,p) \geq \mathsf{roundLow}(y,k,p)$.*

Proof 60 The proof of this is very similar to that of Lemma 7.2. ■

Definition 7.7 (Upper gap) Let W be a finite weight sequence. The *upper gap* of W with discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$, denoted $\mathsf{gapUpper}(W,k,p)$, is

$$\mathsf{gapUpper}(W,k,p) = \begin{cases} 0, \text{ for } |W| = 0 \\ \mathsf{roundUpper}(\mathsf{gapUpper}(U,k,p) + u, k, p) \text{ for } W = U \cdot u \end{cases}$$

Definition 7.8 (Upper approximation of discounted-sum) Let W be a finite weight sequence. The *upper approximation of discounted sum*, called upper DS, for weight sequence W with discount factor $d = 1+2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ is denoted by and defined as

$$\mathsf{DSUpper}(W,k,p) = \mathsf{gapUpper}(W,k,p)/d^{|W|-1}$$

Definition 7.8 is completed by showing the it indeed corresponds to an upper approximation of discounted sum. This requires a basic lemma statement:

Lemma 7.9 *Let $k, p > 0$ be rational-valued parameters. Let $d = 1 + 2^{-k}$ be the non-integer, rational discount factor and $\varepsilon = 2^{-p}$ be the approximation factor. Let W be a finite non-empty weight sequence. Then $0 \leq \mathsf{gapUpper}(W,k,p) - \mathsf{gap}(W,d) < \mathsf{gap}(R_{|W|},d)$.*

Proof 61 The proof argument follows by induction on length of weight sequence W. It makes use of Lemma 7.7 and Lemma 7.8, and closely follows the proof presented in Lemma 7.3. ■

Theorem 7.7 *Let $d = 1 + 2^{-k}$ be the discount factor and $\varepsilon = 2^{-p}$ be the approximation factor, for rationals $p, k > 0$. Then for all weight sequences W, $0 \leq \mathsf{DSUpper}(W, k, p) - DS(W, d) < d \cdot \varepsilon$.*

Proof 62 The proof argument makes use of Lemma 7.9 and closely follows that of Theorem 7.2. ■

7.5.2 Comparator automata for upper approximation of DS

This section constructs a regular comparator for the upper DS defined above. The construction here differs from that of the comparator for lower DS in only one aspect - the values of the thresholds within which it is sufficient to track the value of upper gap in. In this section, we define the comparison language and its comparator automata, prove the necessary thresholds and give the complete construction of the comparator.

Definition 7.9 (Comparison language for upper approximation of DS) Let $\mu > 0$ be an integer bound, and k, p be positive rationals. The *comparison language for upper approximation of discounted sum* with discount factor $d = 1 + 2^{-k}$, approximation factor $\varepsilon = 2^{-p}$, upper bound μ and inequality relation $\mathsf{R} \in \{\leq, \geq\}$ is a language that accepts bounded and finite weight sequence $W \in \Sigma^*$ iff $\mathsf{DSUpper}(W, k, p)$ R 0 holds.

Definition 7.10 (Comparator automata for upper approximation of DS) Let $\mu > 0$ be an integer bound, and k, p be positive rationals. The *comparator*

automata for upper approximation of discounted sum with discount factor $d = 1+2^{-k}$, approximation factor $\varepsilon = 2^{-p}$, upper bound μ and inequality relation $\mathsf{R} \in \{\leq, \geq\}$ is an automaton that accepts the corresponding comparison language.

We establish the range of sufficient values for the upper gap. The new bounds are as follows:

Lemma 7.10 *Let $\mu > 0$ be an integer bound. Let k, p be positive rationals s.t. $d = 1 + 2^{-k}$ is the discount factor, and $\varepsilon = 2^{-p}$ is the approximation factor. Let W be a finite and bounded weight sequence.*

1. *If* $\mathsf{gapUpper}(W, k, p) \leq -\mu \cdot 2^k - 2^{-p}$ *then for all* $u \in \{-\mu, \ldots, \mu\}$, $\mathsf{gapUpper}(W \cdot u, k, p) \leq -\mu \cdot 2^k - 2^{-p}$.

2. *If* $\mathsf{gapUpper}(W, k, p) \geq \mu \cdot 2^k$, *then for all* $u \in \{-\mu, \ldots, \mu\}$, $\mathsf{gapUpper}(W \cdot u, k, p) \geq \mu \cdot 2^k$.

Proof 63 Part 1. Let W and u be as defined above. Then $\mathsf{gapUpper}(W \cdot u, k, p) = \mathsf{roundUpper}(d \cdot \mathsf{gapUpper}(W, k, p) + u, k, p)$. From Lemma 7.7, we get that $\mathsf{gapUpper}(W \cdot u, k, p) \leq d \cdot \mathsf{gapUpper}(W, k, p) + u + 2^{-(p+k)}$. From our assumption, we further get that $\mathsf{gapUpper}(W \cdot u, k, p) \leq d \cdot -(\mu \cdot 2^k + 2^{-p}) + u = (1+2^{-k}) \cdot -(\mu \cdot 2^k + 2^{-p}) + u + 2^{-(p+k)} = -\mu \cdot 2^k - \mu - 2^{-p} - 2^{-(p+k)} + u + 2^{-(p+k)} \leq -(\mu \cdot 2^k + 2^{-p})$.

Part 2. Let W and u be as defined above. Then $\mathsf{gapUpper}(W \cdot u, k, p) = \mathsf{roundUpper}(d \cdot \mathsf{gapUpper}(W, k, p) + u, k, p)$. From Lemma 7.7, we get that $\mathsf{gapUpper}(W \cdot u, k, p) \geq d \cdot \mathsf{gapUpper}(W, k, p) + u$. Further, from our assumptions we get that $\mathsf{gapUpper}(W \cdot u, k, p) \geq d \cdot \mu \cdot 2^k + u = (1 + 2^{-k}) \cdot \mu \cdot 2^k + u = \mu \cdot 2^k + \mu + u \geq \mu \cdot 2^k$, since $\mu \geq u$. ■

Construction

Let $\mu > 0$, $d = 1 + 2^{-k}$, $\varepsilon = 2^{-p}$ be the upper bound, discount factor and approximation factor, respectively. Let T_l be the largest integer such that $\mathsf{T}_l \cdot 2^{-(p+k)} \leq -\mu \cdot 2^k$. Let T_u be the smallest integer such that $\mathsf{T}_u \cdot 2^{-(p+k)} \geq \mu \cdot 2^k + 2^{-p}$. Note, the thresholds are from Lemma 7.10. For relation $\mathsf{R} \in \{\leq, \geq\}$, construct DFA $\mathsf{compUpper}(\mu, k, p, \mathsf{R}) = (S, s_I, \Sigma, \delta, \mathcal{F})$ as follows:

- $S = \{\mathsf{T}_l, \mathsf{T}_l + 1, \ldots, \mathsf{T}_u\}$, $s_I = \{0\}$ and $\mathcal{F} = \{i | i \in S \text{ and } i \ R \ 0\}$
- Alphabet $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu - 1, \mu\}$
- Transition function $\delta \subseteq S \times \Sigma \to S$ where $(s, a, t) \in \delta$ then:
  1. If $s = \mathsf{T}_l$ or $s = \mathsf{T}_u$, then $t = s$ for all $a \in \Sigma$
  2. Else, let $\mathsf{roundUpper}(d \cdot s \cdot 2^{-(p+k)} + a, k, p) = i \cdot 2^{-(p+k)}$ for an integer i
     (a) If $\mathsf{T}_l \leq i \leq \mathsf{T}_u$, then $t = i$
     (b) If $i > \mathsf{T}_h$, then $t = \mathsf{T}_u$
     (c) If $i < \mathsf{T}_l$, then $t = \mathsf{T}_l$

Theorem 7.8 *Let $\mu > 0$ be and integer upper bound. Let $k, p > 0$ be rational parameters s.t. $d = 1 + 2^{-k}$ is the discount factor and $\varepsilon = 2^{-p}$ is the approximation parameter. DFA $\mathsf{compUpper}(\mu, k, p, \mathsf{R})$ accepts a finite weight sequence $W \in \Sigma^*$ iff $\mathsf{DSUpper}(W, k, p)$ R 0. DFA $\mathsf{compUpper}(\mu, k, p, \mathsf{R})$ has $\mathcal{O}(\mu \cdot 2^{2k+p})$ states.*

7.5.3 Algorithm upperApproxDSInc

This section utilizes the comparator for upper approximation of DS to describe upperApproxDSInc (Algorithm 6). Recall, given inputs P, Q, discount factor $1 < d < 2$

and approximation factor $0 < \varepsilon < 1$, $\mathsf{upperApproxDSInc}(P, Q, d, \varepsilon)$ returns $P \subseteq Q$ holds or $P \subseteq Q - d \cdot \varepsilon$ does not hold.

Once again, the intuition and algorithm design fo $\mathsf{upperApproxDSInc}$ resembles that of $\mathsf{lowApproxDSInc}$. Intuitively, $\mathsf{upperApproxDSInc}$ solves whether P is f-included in Q where aggregate function f is the upper approximation of discounted-sum. Lemma 7.11 precisely states the intuition, and the algorithm is given in Algorithm 6.

We begin with formalizing the intuition. We say, a run ρ_P of word w in P is *dominated* by Q if there exists a run ρ_Q in Q on the same word such that $\mathsf{DSUpper}(\rho_P - \rho_Q, k, p) \leq 0$. Then,

Lemma 7.11 *Given DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$.*

1. *If all runs in P are dominated by Q, then $P \subseteq Q$ holds.*
2. *If there exists a run in P that is not dominated by Q, then $P \subseteq Q - d \cdot \varepsilon$ does not hold.*

Proof 64 Proof of (1.) Let for all words $w \in \Sigma^*$, for all runs of w in $\rho_P \in P$, there exists a run of word w $\rho_Q \in Q$ such that $\mathsf{DSUpper}(\rho_P - \rho_Q, k, p) \leq 0$ implies that $DS(\rho_P - \rho_Q, d) \leq 0$. By arguing as in Lemma 7.5, we get that $P \subseteq Q$.

Proof of (2.) Let $w \in \Sigma*$ be a word for which there exists a run of $w\rho_P \in P$ such that for all runs of w $\rho_Q \in Q$, $\mathsf{DSUpper}(\rho_P - \rho_Q, k, p) > 0$ implies $DS(\rho_P - \rho_Q, d) > -d \cdot \varepsilon$. Therefore, by arguing as done in Lemma 7.5, we get that $P \subseteq Q - d \cdot \varepsilon$ does not hold. ■

We design Algorithm 6 so that it resembles Algorithm 5 except that in this case the comparator refers to that of the upper approximation of discounted-sum:

Algorithm 6 upperApproxDSInc(P, Q, d, ε)

Inputs: DS automata P, Q, discount factor $1 < d < 2$, approximation factor $0 < \varepsilon < 1$

1: $\hat{P} \leftarrow$ AugmentWtAndLabel(P)

2: $\hat{P} - Q \leftarrow$ productDif$(\hat{P}, Q)$

3: dominatedWitness $\leftarrow$ Intersect$(\hat{P} - Q,$ compUpper$(\mu, \log \frac{1}{(d-1)}, \log \frac{1}{\varepsilon}, \leq))$ where μ is the maximum of the absolute value of weights in $\hat{P} - Q$

4: dominated $\leftarrow$ Project(dominatedWitness)

5: $\hat{P}_{-wt} \subseteq$ dominated

6: **if** aux **then**

7: **return** $P \subseteq Q =$ True

8: **else**

9: **return** $P \subseteq Q - d \cdot \varepsilon =$ False

10: **end if**

Lemma 7.12 *Given DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$,* upperApproxDSInc(P, Q, d, ε) *returns* True *iff all runs in P are dominated by Q.*

Proof 65 The proof argument is similar to that in Lemma 7.6. ■

Theorem 7.9 *(*Soundness*) For all inputs DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$, algorithm* upperApproxDSInc *is sound.*

Proof 66 The proof is similar to proof that Theorem 7.4. ■

Theorem 7.10 (Complexity) *Given DS automata P and Q, discount factor $d = 1 + 2^{-k}$ and approximation factor $\varepsilon = 2^{-p}$ for rational values $k, p > 0$. Let μ be the*

absolute value of the largest weight in P and Q. Then the worst case complexity of upperApproxDSInc *is* $2^{\mathcal{O}(n)}$ *where* $n = |P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2 \cdot \varepsilon}$.

Proof 67 The proof is similar to that of Theorem 7.5. ■

7.6 Anytime algorithm for DS inclusion

This section describes the core contribution of this work. We design an anytime algorithm for discounted-sum inclusion. On inputs DS automata P and Q, and discount factor $1 < d < 2$, our algorithm DSInclusion(P, Q, d) either terminates and returns a crisp True or False answer to $P \subseteq Q$, or it establishes a $d \cdot \varepsilon$-close approximation, where the approximation factor $\varepsilon > 0$ decreases with time. In addition, algorithm DSInclusion is co-computational enumerable i.e. if $P \subseteq Q$ does not hold then DSInclusion(P, Q, d) is guaranteed to terminate with that outcome after a finite amount of time.

This section proves soundness, and co-computational enumerability of DSInclusion. Finally, we evaluate the complexity of running DSInclusion upto a desired approximation (Theorem 7.13). The analysis reveals that as ε tends to 0, the worst-case complexity grows rapidly.

Algorithm details

DSInclusion invokes anytimeInclusion with an initial approximation factor $0 < \varepsilon_{\text{init}} < 1$, where anytimeInclusion is described in Algorithm 4. Here $\varepsilon_{\text{init}} = \frac{1}{2}$. Formally, DSInclusion$(P, Q, d)$ = anytimeInclusion$(P, Q, d, 0.5)$ Recall, Algorithm 4 is a tail recursive procedure in which subprocedures lowApproxDSInc and upperApproxDSInc are invoked in each round of the recursion. If the current round of recursion is invoked

with approximation factor $\varepsilon > 0$, then lowApproxDSInc and upperApproxDSInc are invoked with ε. In this round, if lowApproxDSInc or upperApproxDSInc returns $P \subseteq Q$ does not hold or $P \subseteq Q$ holds, respectively, anytimeInclusion terminates. Otherwise, it invokes anytimeInclusion with approximation factor $\frac{\varepsilon}{2}$.

In order to analyse DSInclusion, we begin with some useful terminology: DSInclusion(P, Q, d) is said to be in the *ε-th round* when the current invocation of anytimeInclusion occurs with approximation factor ε. DSInclusion(P, Q, d) is said to *terminate in the ε-th round* if anytimeInclusion(P, Q, d, ε) returns a crisp solution to DS inclusion in the ε-th round. Finally, DSInclusion(P, Q, d) is said to *terminate* if there exists a $\varepsilon > 0$ such that it terminates in the ε-th round.

Theorem 7.11 *[Soundness] Let P, Q be DS automata, and $1 < d < 2$ be the discount factor.*

1. *If P is not DS-included in Q and* DSInclusion(P, Q, d) *terminates, then* DSInclusion(P, Q, d) *returns* $P \subseteq Q =$ False

2. *If P is DS-included in Q holds and* DSInclusion(P, Q, d) *terminates, then* DSInclusion(P, Q, d) *returns* $P \subseteq Q =$ True

3. *If* DSInclusion(P, Q, d) *does not terminate in the ε-th round in which the approximation factor is $\varepsilon > 0$, then $P \subseteq Q + d \cdot \varepsilon$ holds.*

Proof 68 We reason about anytimeInclusion as that is sufficient.

Proof of (1.) and (2.): anytimeInclusion terminates only if either lowApproxDSInc returns $P \subseteq Q =$ False or upperApproxDSInc returns $P \subseteq Q =$ True. Since each of these subprocedures is individually sound (Theorem 7.4 and Theorem 7.9, respectively), statements (1.) and (2.) hold.

Proof of (3.): anytimeInclusion does not terminate in the ε-th round only if lowApproxDSInc and upperApproxDSInc return $P \subseteq Q + d \cdot \varepsilon = \mathsf{True}$ and $P \subseteq Q - d \cdot \varepsilon = \mathsf{False}$, respectively. Therefore, by soundness of lowApproxDSInc (Theorem 7.4), it holds that P is $d \cdot \varepsilon$-close to Q. ■

Next, we prove that DSInclusion is co-computational enumerable.

Theorem 7.12 (Co-ce) *Let P, Q be DS automata, and $1 < d < 2$ be the discount factor. If $P \subseteq Q = \mathsf{False}$ then $\mathsf{DSInclusion}(P, Q, d)$ terminates after a finite number of recursions. Upon termination, $\mathsf{DSInclusion}(P, Q, d)$ returns $P \subseteq Q = \mathsf{False}$.*

Proof 69 By soundness of upperApproxDSInc, we know that upperApproxDSInc will return $P \subseteq Q - d \cdot \varepsilon = \mathsf{False}$ in every invocation. Hence, DSInclusion cannot terminate due to upperApproxDSInc. Therefore, if DSInclusion terminates, it must be because lowApproxDSInc returns $P \subseteq Q = \mathsf{False}$ for some $\varepsilon > 0$. It remains to show that such an approximation factor exists. But that is exactly the result in Theorem 7.6. Therefore, if $P \subseteq Q = \mathsf{False}$, then DSInclusion is guaranteed to terminate with the outcome $P \subseteq Q = \mathsf{False}$. ■

Note that we cannot determine, apriori, the number of recursive invocations that will be conducted for a given input instance. If we could, then we could have proved the decidability of discounted sum inclusion for discount factor $1 < d < 2$. As a result, one cannot determine the worst-case complexity of DSInclusion in general. However, the worst-case complexity can be computed upto a certain precision. More precisely, if a user decides that it will run DSInclusion until either it terminates with a crisp solution or $d \cdot \varepsilon_c$-close approximation is established, where approximation factor ε_c is pre-determined. Note that it is not sufficient to recursively only invoke lowApproxDSInc till the approximation factor is ε_c since then we would never be able

to generate the outcome that DS inclusion holds. The worst-case complexity upto ε_c is computed as follows:

Theorem 7.13 (Complexity given precision) *Given DS automata P and Q, discount factor $1 < d < 2$. Let μ be the largest weight in P and Q. Let $0 < \varepsilon_{\mathsf{c}} < 1$ be the desired precision. Then, the worst-case complexity of solving DS inclusion upto a precision of ε_{c} is $2^{\mathcal{O}(n)}$ where $n = |P| \cdot |Q| \cdot \frac{\mu}{(d-1)^2} \cdot \log(\frac{1}{\varepsilon_c})$*

Proof 70 Without loss of generality, let $\varepsilon_c = 2^{-m}$ for$m \in \mathbb{N}$. Then in the worst case, DSInclusion will terminate after invoking anytimeInclusion with ε_c. The worst case complexity of DSInclusion with precision ε_c is calculate by taking the sum of the complexity of each invocation of anytimeInclusion till that point. The calculation evaluates to the expression in the statement. ■

So, as ε_{c} converges to 0, i.e, DS inclusion is solved exactly, the worst-case complexity explodes rapidly. This renders a quantitative measure of the difficulty of solving DS inclusion. Although this isn't concrete evidence for undecidability of DS inclusion, it certainly points in that direction. Finally, if $d = 1$, DS inclusion would be the same as sum. Then the above evaluation corroborates the known undecidability of quantitative inclusion with sum [11].

7.7 Chapter summary

This chapter investigates DS inclusion when the discount factor $1 < d < 2$ is not an integer. The decidability of this problem has been open for more that a decade now. So, this chapter focuses on designing solutions for DS inclusion that could be used in practice despite its decidability being unknown. To this effect, we design an anytime algorithm for DS inclusion. The algorithm may not always solve the problem

exactly. In these cases, it will generate an approximate result, which is a meaningful outcome in practice. To the best of our knowledge, this is the first attempt to solve DS inclusion with non-integer discount factors for practical purposes. While this chapter looks into DS inclusion over finite words, we believe the same ideas can be extended to DS inclusion over infinite words. Our algorithm design is motivated by designing regular comparator automata for approximations of DS with non-integer discount factors. Thus, not only are comparators able to design scalable solutions when the discount factor is an integer, as shown in Chapter 6, they also makes algorithmic advances for non-integer discount factors.

Part III

Quantitative games with discounted-sum

Chapter 8

On the analysis of quantitative games

This part continues the investigation of automata-based quantitative reasoning by studying their impact on quantitative games. We show that even here, comparator automata delivers the benefits of scalability, efficiency, and broader applicability.

8.1 Introduction

Quantitative properties of systems are increasingly being explored in automated reasoning across diverse application areas, including software verification [19, 66, 68], security [43, 56], computer-aided education [53], and even verification of deep neural networks [22, 82]. In decision making domains such as planning and reactive synthesis, quantitative properties have been deployed to obtain high-quality solutions [29], describe hard and soft constraints [69], constraints on cost and resource [58], and the like. In most cases, planning and synthesis with quantitative properties is formulated into an *analysis problem* over a two player, finite-state game arena with a cost model that encodes the quantitative property [37].

Optimization over such games is a popular choice of analysis in literature. Typically, optimization over these games has polynomial time algorithms. However, the degree of polynomial may be too high to scale in practice; resulting in limited applicability of the synthesis task at hand [30]. Furthermore, often times these synthesis tasks are accompanied with temporal goals. Under this extension, the games are

required to generate an optimal solution while also satisfying the temporal objective. However, prior works have proven that optimal strategies may not exist under extension with temporal goals, rendering analysis by optimization incompatible with temporal goals [41].

To this end, we propose an alternate form of analysis in which the objective is to search for a solution that adheres to *a given threshold constraint* as opposed to generating an optimal solution. We call this analysis the *satisficing problem*, a well-established notion that we borrow from economics [1]. Our argument is that in several cases optimal solutions may be replaced with satisficing ones. For instance, a solution with *minimal* battery consumption can be substituted by one that operates *within* the battery life. This work contrasts between the optimization problem and the satisficng problem w.r.t. theoretical complexity, empirical performance, and temporal extensibility.

More specifically, this work studies the aforementioned contrast on two player, finite-state games with the discounted-sum aggregate function as the cost model, which is a staple cost-model in decision making domains [75, 78, 85]. In these games, players take turns to pass a token along the *transition relation* between the states. As the token is pushed around, the play accumulates weights along the transitions using the discounted-sum cost model. The players are assumed to have opposing objectives: one player maximizes the cost while the other minimizes it. The optimization problem is to find the optimal cost of all possible plays in the game [83]. We define the satisficing problem as follows: Given a threshold value $v \in \mathbb{Q}$, does there exist a strategy for the minimizing player such that the cost of all resulting plays is less than (or $\leq$) to the threshold v?

From prior work, one can infer that the optimization problem is pseudo-

polynomial [57, 93]. Zwick and Patterson have shown that a value-iteration (VI) algorithm converges to the optimal cost at infinitum [93]. Interestingly, even though the VI algorithm finds extensive usage [29, 30], a thorough worst-case analysis of VI has hitherto been absent. This work, first of all, amends the oversight. In § 8.2, we present the analysis for VI for all discount factors $d > 1$. Towards this, we observe that VI performs many arithmetic operations. Therefore, it is crucial to account for the cost of arithmetic operations as well. Its significance is emphasised as we show that there are orders of magnitude of difference between the complexity of VI under unit-cost and bit-cost models of arithmetic. For instance, when the discount factor is an integer, we show that VI is $\mathcal{O}(|V|^2) \cdot |E|)$ and $\mathcal{O}(|V|^4 \cdot |E|)$ under unit- and bit-cost model, respectively, where V and E are the set of states and transitions. In addition, we observe that VI can take as many as $\Theta(|V|^2)$ iterations to compute the optimal value. Note that this bound is tight, indicating that the scalability of VI will be limited only to games with a small number of states and transitions. We confirm this though an empirical analysis. We show that despite heuristics, VI-based algorithms cannot escape their worst-case behavior, adversely impacting its empirical performance (§ 8.4). Finally, it is known that these games may not have optimal solutions when extended with temporal goals [41].

In contrast, our examination of the satisficing problem illustrates its advantages over optimization. We solve satisficing via an automata-based approach as opposed to an arithmetic approach (§ 8.3). Our approach is motivated by recent advances in automata-based reasoning of quantitative properties using *comparator automata* [25, 28]. We show that when the discount factor is an integer, satisficing problem can be solved in $\mathcal{O}(|V| + |E|)$ via an efficient reduction to safety/reachability games [87]. Observe that there is a fundamental separation between the complexity of satisficing

and the number of iterations required in VI, indicating a computational gain of our comparator-based solution for satisficing. As before, an empirical evaluation confirms this as well. Last but not the least, we show that unlike optimization, satisficing naturally integrates with temporal goals. The reason is that since both, saisficing and temporal goals, adopt automata-based solutions, the two can be seamlessly combined with one another (§ 8.6). Currently, our method works for integer discount factors only. We believe, these results can be extended to the non-integer case with some approximation guarantee.

8.2 Optimization problem

The optimization problem can be solved by a VI algorithm (§ 8.2.1 [93]). While a reduction from mean-payoff games proves that the VI algorithm is pseudo-polynomial, a thorough worst-case analysis of VI has been missing. This section undertakes a thorough investigation of its worst-case complexity.

Our analysis also exposes the dependence of VI on the discount factor $d > 1$ and the cost-model for arithmetic operations i.e. unit-cost or bit-cost model. We observe that these parameters bring about drastic changes to the algorithm's theoretical evaluation. Our analysis works for all discount factors $d > 1$.

We begin by describing the VI algorithm in § 8.2.1. First, we prove that it is sufficient for VI to perform a finite-number of iterations to compute the optimal value in § 8.2.2. Finally, this result is used to compute the worst-case complexity of VI under the unit- and bit-cost models of arithmetic in § 8.2.3.

8.2.1 Value-iteration algorithm

The VI algorithm plays a min-max game between the two players [93]. Let $wt_k(v)$ denote the optimal cost of a k-length game that begins in state $v \in V$. Then $wt_k(v)$ can be computed using the following equations: The optimal cost of a 1-length game beginning in state $v \in V$ is as follows:

$$wt_1(v) = \begin{cases} max\{\gamma(v,w)|(v,w) \in E\} & \text{if } v \in V_0 \\ min\{\gamma(v,w)|(v,w) \in E\} & \text{if } v \in V_1 \end{cases}$$

Given the optimal-cost of a k-length game, the optimal cost of a $(k+1)$-length game is computed as follows:

$$wt_{k+1}(v) = \begin{cases} max\{\gamma(v,w) + \frac{1}{d} \cdot wt_k(w)|(v,w) \in E\} \text{ if } v \in V_0 \\ min\{\gamma(v,w) + \frac{1}{d} \cdot wt_k(w)|(v,w) \in E\} \text{ if } v \in V_1 \end{cases}$$

Then, it has been shown that the optimal cost of a k-length game beginning at state $v \in V$ converges to the optimal cost of an infinite-length game beginning in state $v \in V$ [83, 93] as $k \to \infty$. In particular, let W be the optimal value (beginning in state v_{init}), then $W = \lim_{\to k \to \infty} wt_k(v_{init})$.

8.2.2 Number of iterations

The VI algorithm, as defined above, waits for convergence to terminate. Clearly, that is not sufficient for a worst-case analysis. To this end, in this section we establish a crisp bound on the number of iterations required to compute the optimal value. Furthermore, we show that the bound we calculate is tight.

Upper bound on number of iterations.

We prove the upper bound in several steps, as described below:

Step 1 We show that the optimal value W must fall between an interval such that as the number of iterations k of the VI algorithm increases, the interval length converges to 0.

Step 2 We show that the optimal value W is a rational number with denominator at most $\mathsf{bound_W}$, where $\mathsf{bound_W}$ is parameterized by numerator and denominator of the discount factor and the number of states in the graph game.

Step 3 Next, we show that the denominator of the minimum non-zero difference between possible values of the optimal cost is at most $\mathsf{bound_{diff}}$, where $\mathsf{bound_{diff}}$ is also parameterized by numerator and denominator of the discount factor and the number of states in the graph game.

Step 4 Finally, we use the previous three Steps to prove the pseudo-polynomial bound. Since the interval in Step 1 converges to 0, we can choose a k such that the interval is less than than $1/\mathsf{bound_{diff}}$. In our computations below, we will see that $\frac{1}{\mathsf{bound_{diff}}} < \frac{1}{\mathsf{bound_v}}$. Therefore, there can be only one rational number with denominator $\mathsf{bound_W}$ or less in the interval identified by the chosen k. Since this interval must also contain the optimal value W, the unique rational number with denominator less than or equal to $\mathsf{bound_W}$ must be the optimal value W.

Our task is to compute the value of k.

We prove all of these steps one-by-one. We begin with proof of Step 1. We show that there is a finite-horizon approximation of the optimal value, i.e., for all $k \in \mathbb{N}$ the optimal value can be bounded using $wt_k(v_{init})$ as follows:

Lemma 8.1 *Let W be the optimal value of a graph game G. Let $\mu > 0$ be the*

maximum of absolute value of cost on all transitions in G. Then, for all $k \in \mathbb{N}$,

$$wt_k(v_{\mathsf{init}}) - \frac{1}{d^{k-1}} \cdot \frac{\mu}{d-1} \leq W \leq wt_k(v_{\mathsf{init}}) + \frac{1}{d^{k-1}} \cdot \frac{\mu}{d-1}$$

Proof 71 This holds because since W is the limit of $wt_k(v_{\mathsf{init}})$ as $k \to \infty$, its value must lie in between the minimum and maximum cost possible if the k-length game is extended to an infinite-length game. The minimum possible extension would be when the k-length game is extended by iterations in which the cost incurred in each round is $-\mu$. Therefore, the resulting lowest value is $wt_k(v_{\mathsf{init}}) - \frac{1}{d^{k-1}} \cdot \frac{\mu}{d-1}$. Similarly, the maximum value is $wt_k(v_{\mathsf{init}}) + \frac{1}{d^{k-1}} \cdot \frac{\mu}{d-1}$. ■

Clearly, as $k \to \infty$, the interval around the optimal value W converges to 0. This way we can find an arbitrarily small interval around the optimal value W.

To prove Step 2, we know that there exist memoryless optimal strategies for both players [83]. Therefore, there must exists an optimal play that is in the form of a *simple lasso*. A *lasso* is a play represented as $v_0 v_1 \dots v_n (s_0 s_2 \dots s_m)^\omega$. The initial segment $v_0 v_1 \dots v_n$ is called the *head* of the lasso, and the cycle segment $s_0 s_1 \dots s_m$ the *loop* of the lasso. A lasso is *simple* if each state in $\{v_0 \dots v_n, s_0, \dots s_m\}$ is distinct. Hence, in order to compute the optimal value, it is sufficient to examine simple lassos in the game only. Therefore, we evaluate the DS of cost sequences derived from simple lassos only. Let $l = a_0 \dots a_n (b_0 \dots b_m)^\omega$ be the cost sequence of a lasso. Let $l_1 = a_0 \dots a_n$ and $l_2 = b_0 \dots b_m$ correspond to the cost sequences of the head and loop of lasso, respectively. Then,

Lemma 8.2 *Let $l = l_1 \cdot (l_2)^\omega$ represent an integer cost sequence of a lasso, where l_1 and l_2 are the cost sequences of the head and loop of the lasso. Let $d = \frac{p}{q}$ be the discount factor. Then, $DS(l, d)$ is a rational number with denominator at most $(p^{|l_2|} - q^{|l_2|}) \cdot (p^{|l_1|})$.*

Proof 72 The result can proven by unrolling the expression of DS of a cost-sequence on a lasso path. The discounted sum of l is given as follows:

$$\begin{aligned}
DS(l,d) =& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot (DS((l_2)^\omega, d)) \\
=& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \left(DS(l_2,d) + \frac{1}{d^{|l_2|}} \cdot DS(l_2,d) + \frac{1}{d^{2\cdot|l_2|}} \cdot DS(l_2,d) + \dots \right) \\
& \text{Taking closed form expression of the term in the parenthesis, we get} \\
=& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \left(\frac{d^{|l_2|}}{d^{|l_2|}-1} \right) \cdot DS(l_2,d) \\
& \text{Let } l_2 = b_0 b_1 \dots b_{|l_2|-1} \text{ where } b_i \in \mathbb{Z} \\
=& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \left(\frac{d^{|l_2|}}{d^{|l_2|}-1} \right) \cdot \left(b_0 + \frac{b_1}{d} + \dots + \frac{b_{|l_2|-1}}{d^{|l_2|-1}} \right) \\
=& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \left(\frac{1}{d^{|l_2|}-1} \right) \cdot \left(b_0 \cdot d^{|l_2|} + b_1 \cdot d^{|l_2|-1} + \dots + b_{|l_2|-1} d \right) \\
& \text{Expressing } d = \frac{p}{q}, \text{ we get} \\
=& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \frac{q^{|l_2|}}{p^{|l_2|}-q^{|l_2|}} \cdot \left(b_0 (\frac{p}{q})^{|l_2|} + \dots b_{|l_2|-1} \cdot \frac{p}{q} \right) \\
& DS(l_1,d) + \frac{1}{d^{|l_1|}} \cdot \frac{1}{p^{|l_2|}-q^{|l_2|}} \cdot M, \text{ where } M \in \mathbb{Z} \\
& \text{Expressing } d = \frac{p}{q} \text{ again, we get} \\
=& \frac{1}{p^{|l_1|}} \cdot \frac{1}{p^{|l_2|}-q^{|l_2|}} \cdot N, \text{ where } N \in \mathbb{Z}
\end{aligned}$$

■

The essence of Lemma 8.2 is that the DS of the cost-sequence of a lasso, simple or non-simple, is a rational number. From here, we can immediately derive Step 2:

Corollary 8.1 *Let $G = (V, v_{init}, E, \gamma)$ be a graph game. Let $d = \frac{p}{q}$ be the discount factor. Then the optimal value of the game is a rational number with denominator at most $(p^{|V|} - q^{|V|}) \cdot (p^{|V|})$*

Proof 73 The optimal value is obtained on a simple lasso as both players have

memoryless strategies that result in the optimal value. The length of the simple lasso is at most $|V|$. Therefore, the length of the head and loop are at most $|V|$ each. Hence, the expression from Lemma 8.2 simplifies to $(p^{|V|} - q^{|V|}) \cdot (p^{|V|})$. ■

Similarly, Step 3 is resolved as follows:

Corollary 8.2 *Let $G = (V, v_{init}, E, \gamma)$ be a graph game. Let $d = \frac{p}{q}$ be the discount factor. Then the minimal non-zero difference between the cost on simple lassos is a rational number with denominator at most $(p^{(|V|^2)} - q^{(|V|^2)}) \cdot (p^{(|V|^2)})$.*

Proof 74 The difference of two lassos l_1 and l_2 can be represented by another lasso $l = l_1 \times l_2$ constructed from taking their product, and assigning the difference of their costs on each transition. If the maximum length of the lassos is $|V|$, then the maximum length of the difference lasso will be $|V|^2$. Then, from Lemma 8.2 we immediately obtain that the upper bound of the denominator of the minimum non-zero difference of optimal plays is $(p^{(|V|^2)} - q^{(|V|^2)}) \cdot (p^{(|V|^2)})$. ■

Therefore, $\mathsf{bound}_W = (p^{|V|} - q^{|V|}) \cdot (p^{|V|})$ and $\mathsf{bound}_{\mathsf{diff}} = (p^{(|V|^2)} - q^{(|V|^2)}) \cdot (p^{(|V|^2)})$. Recall, in [93] Zwick-Paterson *informally claim* that a pseudo-polynomial algorithm can be devised to compute the optimal value of the game. We formalize their statement in the final step as follows:

Theorem 8.1 *Let $G = (V, v_{init}, E, \gamma)$ be a graph game. The number of iterations required by the value-iteration algorithm or the length of the finite-length game to compute the optimal value W is*

1. *$\mathcal{O}(|V|^2)$ when discount factor $d \geq 2$,*
2. *$\mathcal{O}\left(\frac{\log(\mu)}{d-1} + |V|^2\right)$ when discount factor $1 < d < 2$.*

Proof 75 Recall, the task is to find a k such that the interval identified by Step 1 is less than $\frac{1}{\mathsf{bound}_{\mathsf{diff}}}$. Note that $\mathsf{bound}_W < \mathsf{bound}_{\mathsf{diff}}$. Therefore, $\frac{1}{\mathsf{bound}_{\mathsf{diff}}} < \frac{1}{\mathsf{bound}_{\mathsf{v}}}$. Hence, there can be only one rational value with denominator $\mathsf{bound}_{\mathsf{W}}$ or less in the small interval identified by the chosen k. Since the optimal value must also lie in this interval, the unique rational number with denominator $\mathsf{bound}_{\mathsf{W}}$ or less must be the optimal value. Let k be such that the interval from Step 1 is less than $\frac{1}{\mathsf{bound}_{\mathsf{diff}}}$. Then,

$$
\begin{aligned}
2 \cdot \frac{\mu}{d-1 \cdot d^{k-1}} &\leq c \cdot \frac{1}{(p^{(|V|^2)} - q^{(|V|^2)}) \cdot (p^{(|V|^2)})} \text{ for some } c > 0 \\
2 \cdot \frac{\mu}{d-1 \cdot d^{k-1}} &\leq c \cdot \frac{q^{2 \cdot |V|^2}}{(p^{(|V|^2)} - q^{(|V|^2)}) \cdot (p^{(|V|^2)})} \text{ for some } c > 0 \\
2 \cdot \frac{\mu}{d-1 \cdot d^{k-1}} &\leq c \cdot \frac{1}{(d^{(|V|^2)} - 1) \cdot (d^{(|V|^2)})} \text{ for } c > 0 \\
d - 1 \cdot d^{k-1} &\geq c' \cdot \mu \cdot (d^{(|V|^2)} - 1) \cdot (d^{(|V|^2)}) \text{ for } c' > 0 \\
2 \cdot \log(d-1) + k \cdot \log(d) &\geq c'' + \log(\mu) + \log(d^{(|V|^2)} - 1) + |V|^2 \cdot \log(d) \text{ for } c'' > 0
\end{aligned}
$$

The following cases occur depending how large or small the values are:

Case 1. When $d \geq 2$: In this case, both d and $d^{|V|^2}$ are large. Then,

$$
\begin{aligned}
2 \cdot \log(d-1) + k \cdot \log(d) &\geq c'' + \log(\mu) + \log(d^{(|V|^2)} - 1) + |V|^2 \cdot \log(d) \text{ for } c'' > 0 \\
2 \cdot \log(d) + k \cdot \log(d) &\geq c'' + \log(\mu) + (|V|^2) \log(d) + |V|^2 \cdot \log(d) \text{ for } c'' > 0 \\
k &= \mathcal{O}(|V|^2)
\end{aligned}
$$

Case 2. When d is small but $d^{|V|^2}$ is not: In this case, $\log(d) \approx (d-1)$, and $\log(d-1) \approx 2 - d$. Then,

$$
\begin{aligned}
2 \cdot \log(d-1) + k \cdot \log(d) &\geq c'' + \log(\mu) + \log(d^{(|V|^2)} - 1) + |V|^2 \cdot \log(d) \text{ for } c'' > 0 \\
2 \cdot \log(d-1) + k \cdot \log(d) &\geq c'' + \log(\mu) + (|V|^2) \log(d) + |V|^2 \cdot \log(d) \text{ for } c'' > 0 \\
2 \cdot (2-d) + k \cdot (d-1) &\geq c'' + \log(\mu) + (|V|^2)(d-1) + |V|^2 \cdot (d-1) \text{ for } c'' > 0 \\
k &= \mathcal{O}\Big(\frac{\log(\mu)}{d-1} + |V|^2\Big)
\end{aligned}
$$

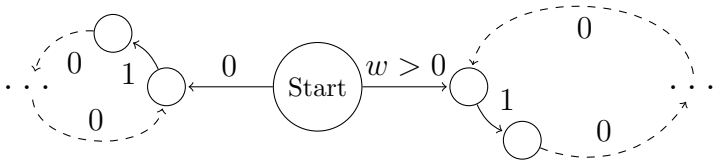


Figure 8.1 : Sketch of game graph which requires $\Omega(|V|^2)$ iterations

Case 3. When both d and $d^{|V|^2}$ are small. Then, in addition to the approximations from the earlier case, $\log(d^{|V|} - 1) \approx (2 - d^{|V|})$. So,

$$2 \cdot \log(d-1) + k \cdot \log(d) \geq c'' + \log(\mu) + \log(d^{(|V|^2)} - 1) + |V|^2 \cdot \log(d) \text{ for } c'' > 0$$

$$2 \cdot \log(d-1) + k \cdot \log(d) \geq c'' + \log(\mu) + 2 - d^{(|V|^2)} + |V|^2 \cdot \log(d) \text{ for } c'' > 0$$

$$2 \cdot (2-d) + k \cdot (d-1) \geq c'' + \log(\mu) + 2 - d^{(|V|^2)} + |V|^2 \cdot (d-1) \text{ for } c'' > 0$$

$$k = \mathcal{O}\Big(\frac{\log(\mu)}{d-1} + |V|^2\Big)$$

■

Lower bound on number of iterations

This bound is tight. We show this by constructing a quantitative graph game for which it is necessary that the VI algorithm takes $\Omega(|V|^2)$ iterations. We give a sketch of this input instance in Fig 8.1. Let all states in Fig 8.1 belong to the maximizing player. Hence, the optimization problem reduces to searching for a *path* with optimal cost. The idea is to show that the cost of finite-length game with lesser than $\Omega(|V|^2)$-length will result in an incorrect optimal cost. Therefore, Fig 8.1 is designed in a way so that the path for optimal cost of a k-length game is along the right hand side (RHS) loop when k is small, but along the left hand side (LHS) loop when k is large. This way, the correct maximal value can be obtained only at a large value for

k. Hence the VI algorithm runs for at least k iterations.

Fig 8.1 realizes these objectives by making the loop to the RHS larger than the one on the LHS, as shown in Fig 8.1, and assigning $w > 0$. The intuition is that when k is small, the optimal path is along the RHS loop since $w > 0$. However, since the RHS is larger, it accumulates cost slower than the LHS loop. As a result, as k becomes larger, for an appropriate assignment to w, the cost on the LHS will eventually be larger than that on the PHS. By meticulous reverse engineering of the size of both loops and the value of w, one can guarantee that $k = \Omega(|V|^2)$.

A concrete instance is as follows: Let the left hand side loop have $4n$ edges, the right hand side of the loop have $2n$ edges, and $w = \frac{1}{d^{3n}} + \frac{1}{d^{7n}} + \cdots + \frac{1}{d^{m \cdot n - 1}}$ such that $m \cdot n - 1 = c \cdot n^2$ for a positive integer $c > 0$. One can show for a finite games of length $(m \cdot n - 1)$ or less, the optimal path arises from the loop to the right. But for games of length greater than $(m \cdot n - 1)$, the optimal path will be to due to the left hand side loop.

8.2.3 Worst-case complexity analysis

Finally, we present the complete worst-case complexity analysis. Since, VI algorithm is dominated by arithmetic operations, we take into account their cost as well. We work with the unit-cost and bit-cost models of arithmetic. We observe that this choice has a drastic impact to the worst-case complexities.

Unit-cost model

Under the unit-cost model of arithmetic, all arithmetic operations are assumed to take constant time.

Theorem 8.2 *Let $G = (V, v_{init}, E, \gamma)$ be a quantitative graph game. The worst-case complexity of computing the optimal value under unit-cost model for arithmetic operations is*

1. $\mathcal{O}(|V|^2 \cdot |E|)$ *when discount factor* $d \geq 2$,
2. $\mathcal{O}\Big(\frac{\log(\mu)\cdot|E|}{d-1} + |V|^2 \cdot |E|\Big)$ *when discount factor* $1 < d < 2$.

Proof 76 The cost of updating the cost of all vertices from iteration j to $j+1$ is linear in $|E|$ since every transition is traversed exactly once. Therefore, the worst-case complexity of computing the optimal values of a k-length game is $\mathcal{O}(k \cdot |E|)$. In particular, the worst-case complexity of computing the optimal value is $\mathcal{O}(|V|^2 \cdot |E|)$ or $\mathcal{O}\Big(\frac{\log(\mu)\cdot|E|}{d-1} + |V|^2 \cdot |E|\Big)$ depending on whether the discount factor $d \geq 2$ or $1 < d < 2$, respectively (Theorem 8.1). ■

Bit-cost model

Under the bit-cost model, the cost of arithmetic operations depends on the size of the numerical values. Iintegers are represented in their bit-wise representation. Rational numbers $\frac{r}{s}$ are represented as a tuple of the bit-wise representation of integers r and s. For two integers of length n and m, the cost of their addition and multiplication is $O(m+n)$ and $O(m \cdot n)$, respectively.

To compute the cost of arithmetic in each iteration of the value-iteration algorithm, we define the cost of a transition $(v, w) \in E$ in the k-th iteration as

$$\mathsf{cost}_1(v, w) = \gamma(v, w) \text{ and } \mathsf{cost}_k(v, w) = \gamma(v, w) + \frac{1}{d} \cdot wt_{k-1}(v) \text{ for } k > 1$$

Then, clearly, $wt_k(v) = \mu\{\mathsf{cost}_k(v, w) | w \in vE\}$ if $v \in V_0$ and $wt_k(v) = \min\{\mathsf{cost}_k(v, w) | w \in vE\}$ if $v \in V_1$. Since, we compute the cost of every transition in each iteration, it is crucial to analyze the size and cost of computing cost.

Lemma 8.3 *Let G be a quantiative graph game. Let $\mu > 0$ be the maximum of absolute value of all costs along transitions. Let $d = \frac{p}{q}$ be the discount factor. Then for all $(v, w) \in E$, for all $k > 0$*

$$\mathsf{cost}_k(v, w) = \frac{q^{k-1} \cdot n_1 + q^{k-2}p \cdot n_2 + \cdots + p^{k-1}n_k}{p^{k-1}}$$

where $n_i \in \mathbb{Z}$ such that $|n_i| \leq \mu$ for all $i \in \{1, \ldots, k\}$.

Proof 77 Lemma 8.3 can be proven by induction on k. ■

Lemma 8.4 *Let G be a quantiative graph game. Let $\mu > 0$ be the maximum of absolute value of all costs along transitions. Let $d = \frac{p}{q}$ be the discount factor. For all $(v, w) \in E$, for all $k > 0$ the cost of computing $\mathsf{cost}_k(v, w)$ in the k-th iteration is $\mathcal{O}(k \cdot \log p \cdot \mu\{\log \mu, \log p\})$.*

Proof 78 We compute the cost of computing $\mathsf{cost}_k(v, w)$ given that optimal costs have been computed for the $(k-1)$-th iteration. Recall,

$$\begin{aligned}\mathsf{cost}_k(v, w) &= \gamma(v, w) + \frac{1}{d} \cdot wt_{k-1}(v) \;=\; \gamma(v, w) + \frac{q}{p} \cdot wt_{k-1}(v) \\ &= \gamma(v, w) + \frac{q}{p} \cdot \frac{q^{k-2} \cdot n_1 + q^{k-3}p \cdot n_2 + \cdots + p^{k-2}n_{k-1}}{p^{k-2}}\end{aligned}$$

for some $n_i \in \mathbb{Z}$ such that $|n_i| \leq \mu$. Therefore, computation of $\mathsf{cost}_k(v, w)$ involves four operations:

1. Multiplication of q with $(q^{k-2} \cdot n_1 + q^{k-3}p \cdot n_2 + \cdots + p^{k-2}n_{k-1})$. The later is bounded by $(k-1) \cdot \mu \cdot p^{k-1}$ since $|n_i| \leq \mu$ and $p > q$. The cost of this operation is $\mathcal{O}(\log((k-1) \cdot \mu \cdot p^{k-1}) \cdot \log(p)) = \mathcal{O}(((k-1) \cdot \log p + \log \mu + \log(k-1)) \cdot (\log p))$.

2. Multiplication of p with p^{k-2}. Its cost is $\mathcal{O}((k-2) \cdot (\log p)^2)$.

3. Multiplication of p^{k-1} with $\gamma(v, w)$. Its cost is $\mathcal{O}((k-1) \cdot \log p \cdot \log \mu)$.

4. Addition of $\gamma(v,w) \cdot p^{k-1}$ with $q \cdot (q^{k-2} \cdot n_1 + q^{k-3}p \cdot n_2 + \cdots + p^{k-2}n_{k-1})$. The cost is linear in their representations.

Therefore, the cost of computing $\mathsf{cost}_k(v,w)$ is $\mathcal{O}(k \cdot \log p \cdot \mu\{\log \mu, \log p\})$. ■

Now, we can compute the cost of computing optimal costs in the k-th iteration from the $k-1$-th iteration.

Lemma 8.5 *Let G be a quantiative graph game. Let $\mu > 0$ be the maximum of absolute value of all costs along transitions. Let $d = \frac{p}{q}$ be the discount factor. The worst-case complexity of computing optimal costs in the k-th iteration from the $(k-1)$-th iteration is $\mathcal{O}(|E| \cdot k \cdot \log \mu \cdot \log p)$.*

Proof 79 The update requires us to first compute the transition cost in the k-th iteration for every transition in the game. Lemma 8.4 gives the cost of computing the transition cost of one transition. Therefore, the worst-case complexity of computing transition cost for all transitions is $\mathcal{O}(|E| \cdot k \cdot \log p \cdot \mu\{\log \mu, \log p\})$.

To compute the optimal cost for each state, we are required to compute the maximum transition cost of all outgoing transitions from the state. Since the denominator is same, the maximum value can be computed via lexicographic comparison of the numerators on all transitions. Therefore, the cost of computing maximum for all states is $\mathcal{O}(|E| \cdot k \cdot \log \mu \cdot \log p)$.

Therefore, total cost of computing optimal costs in the k-th iteration from the $(k-1)$-th iteration is $\mathcal{O}(|E| \cdot k \cdot \log p \cdot \mu\{\log \mu, \log p\})$. ■

Finally, the worst-case complexity of computing the optimal value of the quantitative game under bit-cost model for arithmetic operations is as follows:

Theorem 8.3 *Let $G = (V, v_{init}, E, \gamma)$ be a quantitative graph game. Let $\mu > 0$ be the maximum of absolute value of all costs along transitions. Let $d = \frac{p}{q} > 1$ be the discount factor. The worst-case complexity of computing the optimal value under bit-cost model for arithmetic operations is*

1. $\mathcal{O}(|V|^4 \cdot |E| \cdot \log p \cdot \mu\{\log \mu, \log p\})$ *when* $d \geq 2$,
2. $\mathcal{O}(\left(\frac{\log(\mu)}{d-1} + |V|^2\right)^2 \cdot |E| \cdot \log p \cdot \mu\{\log \mu, \log p\})$ *when* $1 < d < 2$.

Proof 80 This is the sum of computing the optimal costs for all iterations.

When $d \geq 2$, it is sufficient to perform value iteration for $O(|V|^2)$ times (Theorem 8.1). So, the cost is $\mathcal{O}((1 + 2 + 3 \cdot + |V|^2) \cdot |E| \cdot \log p \cdot \mu\{\log \mu, \log p\})$. This expression simplifies to $\mathcal{O}(|V|^4 \cdot |E| \cdot \log p \cdot \mu\{\log \mu, \log p\})$.

A similar computation solves the case for $1 < d < 2$. ■

Final remarks (Integer discount factor)

Our analysis from Theorem 8.1 till Theorem 8.3 show that VI will not scale well despite being polynomial in size of the graph game. Even when the discount factor $d > 1$ is an integer ($d \geq 2$), the algorithm requires $\Theta(|V|^2)$ iterations as Theorem 8.1 is tight. In this case, VI will be $\mathcal{O}(|V|^2 \cdot |E|)$ and $\mathcal{O}(|V|^4 \cdot |E|)$ under the unit-cost and bit-cost models for arithmetic operation, respectively (Theorem 8.2 and Theorem 8.3, respectively).

From a practical point of view, implementations of VI will use the bit-cost model as they may rely on multi-precision libraries in order to to avoid floating-point errors. Therefore, VI for optimization will be expensive in practice. One may argue that the upper bounds from Theorem 8.3 may be tightened. But it must be noted that since

VI requires $\Omega(|V|^2)$ iterations, even a tighter analysis will not significantly improve its performance in practice.

8.3 Satisficing problem

This section formally defines and investigates the *satisficing problem*. The intuition captured by satisficing is to determine whether a player can guarantee that the cost of all plays will never exceeds a given threshold value. Hence, satisficing can be perceived as a decision variant of optimization. In this section, we prove that when the discount factor is an integer, the satisficing problem can be solved in *linear* time in size of the game graph. Therefore, showing that satisficing is more scalable and efficient than optimization. A key feature of our solution for satisficing is that our solution relies on purely automata-based methods and avoids numerical computations.

We begin by formally defining the satisficing problem as follows:

Definition 8.1 (Satisficing problem) Given a quantitative graph game G and a threshold value $v \in \mathbb{Q}$, the *satisficing problem* is to determine whether player P_1 has a strategy such that for all possible resulting plays of the game, the cost of all plays is less than (or $\leq$) to the threshold v, assuming that the objectives of P_0 and P_1 are to maximize and minimize the cost of plays, respectively.

This section is divided into two parts. § 8.3.1 describes the core technique that our solution builds on. At the heart of our solution lie *DS comparator automata*. Prior work on DS comparator automata have been limited to representing languages that accept a bounded weight sequence A iff $DS(A, d)$ inc 0. This section generalizes the definition to accept weight sequence A iff $DS(A, d)$ inc v, for an *arbitrary but fixed threshold* $v \in \mathbb{Q}$. § 8.3.2 presents our complete solution. Here we use the

DS comparators to reduce the satisficing problem to solving a safety or reachability game, when the discount factor is an integer. In this way, we solve satisficing with automata-based methods only.

8.3.1 Foundations of DS comparator automata with threshold $v \in \mathbb{Q}$

This section generalizes DS comparison languages and DS comparator automata to arbitrary rational threshold values $v \in \mathbb{Q}$. It formally defines the notions, and studies their safety or co-safety, and ω-regular properties.

We begin with formal definitions:

Definition 8.2 (DS comparison language with threshold $v \in \mathbb{Q}$) For an integer upper bound $\mu > 0$, discount factor $d > 1$, equality or inequality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, and a threshold value $v \in \mathbb{Q}$ the *DS comparison language with upper bound μ, relation R, discount factor d and threshold value v* is a language of infinite words over the alphabet $\Sigma = \{-\mu, \ldots, \mu\}$ that accepts $A \in \Sigma^\omega$ iff $DS(A, d)$ R v holds.

Definition 8.3 (DS comparator automata with threshold $v \in \mathbb{Q}$) For an integer upper bound $\mu > 0$, discount factor $d > 1$, equality or inequality relation $\mathsf{R} \in \{<, >, \leq, \geq, =, \neq\}$, and a threshold value $v \in \mathbb{Q}$ the *DS comparator automata with upper bound μ, relation R, discount factor d and threshold value v* is an automaton that accepts the DS comparison language with upper bound μ, relation R, discount factor d and threshold value v.

For sake of succinctness, we refer to the DS comparison language and DS comparator automata by *comparison language* and *comparator*, respectively.

Safety and co-safety of DS comparison languages

Our finding here is all DS comparison languages are either safety langauges or co-safety langagues. More interestingly, the only parameter that decides whether a comparison langauge is safety/co-safety is the equality or inequality relation inc. The values of the discount factor and threshold have no implication on this property.

These observations are formally proven next: Recall definitions of safety/co-safety languages and bad-prefixes from Chapter 6.

Theorem 8.4 *Let $\mu > 1$ be the integer upper bound. For arbitrary discount factor $d > 1$ and threshold value v*

1. *DS comparison languages are safety languages for relations $R \in \{\leq, \geq, =\}$.*

2. *DS comparison language are co-safety languages for relations $R \in \{<, >, \neq\}$.*

Proof 81 Due to duality of safety/co-safety languages, it is sufficient to show that DS-comparison language with $\leq$ is a safety language.

Let us assume that DS-comparison language with $\leq$ is not a safety language. Let W be a weight-sequence in the complement of DS-comparison language with $\leq$ such that it does not have a bad prefix.

Since W is in the complement of DS-comparison language with $\leq$, $DS(W, d) > v$. By assumption, every i-length prefix $W[i]$ of W can be extended to a bounded weight-sequence $W[i] \cdot Y^i$ such that $DS(W[i] \cdot Y^i, d) \leq v$.

Note that $DS(W, d) = DS(W[i], d) + \frac{1}{d^i} \cdot DS(W[i \ldots], d)$, and $DS(W[i] \cdot Y^i, d) = DS(W[i], d) + \frac{1}{d^i} \cdot DS(Y^i, d)$. The contribution of tail sequences $W[i \ldots]$ and Y^i to the discounted-sum of W and $W[i] \cdot Y^i$, respectively diminishes exponentially as the value of i increases. In addition, since and W and $W[i] \cdot Y^i$ share a common i-length

prefix $W[i]$, their discounted-sum values must converge to each other. The discounted sum of W is fixed and greater than v, due to convergence there must be a $k \geq 0$ such that $DS(W[k] \cdot Y^k, d) > v$. Contradiction. Therefore, DS-comparison language with $\leq$ is a safety language.

The above intuition is formalized below:

Since $DS(W, d) > v$ and $DS(W[i] \cdot Y^i, d) \leq v$, the difference $DS(W, d) - DS(W[i] \cdot Y^i, d) > 0$.

By expansion of each term, we get $DS(W, d) - DS(W[i] \cdot Y^i, d) = \frac{1}{d^i}(DS(W[i \ldots], d) - DS(Y^i, d)) \leq \frac{1}{d^i} \cdot (\mod DS(W[i \ldots], d) + \mod DS(Y^i, d))$. Since the maximum value of discounted-sum of sequences bounded by μ is $\frac{\mu \cdot d}{d-1}$, we also get that $DS(W, d) - DS(W[i] \cdot Y^i, d) \leq 2 \cdot \frac{1}{d^i} \mod \frac{\mu \cdot d}{d-1}$.

Putting it all together, for all $i \geq 0$ we get

$$0 < DS(W, d) - DS(W[i] \cdot Y^i, d) \leq 2 \cdot \frac{1}{d^i} \mod \frac{\mu \cdot d}{d-1}$$

As $i \to \infty$, $2 \cdot \mod \frac{1}{d^{i-1}} \cdot \frac{\mu}{d-1} \to 0$. So, $\lim_{\to i \to \infty}(DS(W, d) - DS(W[i] \cdot Y^i, d)) = 0$. Since $DS(W, d)$ is fixed, $\lim_{\to i \to \infty} DS(W[i] \cdot Y^i, d) = DS(W, d)$.

By definition of convergence, there exists an index $k \geq 0$ such that $DS(W[k] \cdot Y^k, d)$ falls within the $\frac{\mod DS(W,d)}{2}$ neighborhood of $DS(W, d)$. Finally since $DS(W, d) > 0$, $DS(W[k] \cdot Y^k, d) > 0$ as well. But this contradicts our assumption that for all $i \geq 0$, $DS(W[i] \cdot Y^i, d) \leq 0$.

Therefore, DS-comparator with $\leq$ is a safety comparator.

■

ω-regularity of DS comparison languages

Next, we determine ω-regularity of comparison languages with threshold value $v \in \mathbb{Q}$. The critical parameter in this case is the discount factor $d > 1$. We observe that a

comparison language is ω-regular iff the discount factor is an integer. Finally, while the threshold value does not affect the ω-regularity of a comparison language, it impacts the size of the ω-regular comparator automata (when discount factor $d > 1$ is an integer).

First, we introduce notation. Since $v \in \mathbb{Q}$, w.l.o.g. let us assume that the threshold value is represented by the regular expression $v = v[0]v[1]\dots v[m](v[m+1]v[m+2]\dots v[n])^\omega$. By abuse of notation, we denote both the regular expression $v[0]v[1]\dots v[m](v[m+1]v[m+2]\dots v[n])^\omega$ and the value $DS(v,d)$ by v.

Our first result is that a comparison language with $v \in \mathbb{Q}$ is not ω-regular if the discount factor is not an integer. The result follows immediately from prior work as it is known that comparison language with $v = 0$ is not ω-regular if the discount factor is not an integer [26]. Therefore:

Theorem 8.5 *Let $\mu > 0$ be the integer upper bound, $v \in \mathbb{Q}$ be the threshold value, and $\mathsf{inc} \in \{<, >, \leq, \geq, =, \neq\}$ be an equality or inequality relation. Let the discount factor $d > 1$ be a non-integer. Then, the DS comparison language with μ, d, inc, and v is not ω-regular.*

Next, we prove that DS comparison languages are ω-regular if the discount factor $d > 1$ is an integer. To prove this, we construct the Büchi automaton corresponding to the language. Since Büchi automata are closed under complementation, intersection and union, it is sufficient to construct the (Büchi) comparator automata for one equality or inequality relation inc. We do so for the relation $\leq$. The construction technique resembles the construction of comparator automata for threshold $v = 0$ for inequality relation $\leq$ presented in [28]. Due to space constraints, we present critical lemma statements, theorem statement and high-level intuition. Please refer to [28]

or the supplemental material for details.

Since we know that the comparison language for $\leq$ is a safety language, we begin by characterizing the bad-prefixes of the comparison language with threshold $v \in \mathbb{Q}$ and relation $\leq$. For this, we introduce notation. Let W be a *finite* weight-sequence. By abuse of notation, the discounted-sum of finite-sequence W with discount-factor d is defined as $DS(W,d) = DS(W \cdot 0^\omega, d)$. The *recoverable-gap* of a finite weight-sequences W with discount factor d, denoted $\mathsf{gap}(W,d)$, is its normalized discounted-sum: If $W = \varepsilon$ (the empty sequence), $\mathsf{gap}(\varepsilon, d) = 0$, and $\mathsf{gap}(W,d) = d^{|W|-1} \cdot DS(W,d)$ otherwise [31].

Lemma 8.6 *Let $\mu > 0$ be the integer upper bound, $d > 1$ be an integer discount factor, and the relation* inc *be the inequality $\leq$. Let $v \in \mathbb{Q}$ be the threshold value such that $v = v[0]v[1]\ldots v[m](v[m+1]v[m+2]\ldots v[n])^\omega$. Let W be a non-empty, bounded, finite weight-sequence. Then, weight sequence W is a bad-prefix of the DS comparison language with μ, d, $\leq$ and v iff $\mathsf{gap}(W - v[|W|], d) > \frac{1}{d} \cdot DS(\mathsf{post}_v(|W|), d) + \frac{\mu}{d-1}$.*

Proof 82 Let W be a bad prefix. Then for all infinite length, bounded weight sequence Y we get that $DS(W \cdot Y, d) > v \implies DS(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) \geq DS(v[|W|] \cdot \mathsf{post}_v(|W|), d) \implies DS(W,d) - DS(v[|W|],d) > \frac{1}{d^{|W|}} \cdot (DS(\mathsf{post}_v(|W|),d) - DS(Y,d)) \implies \mathsf{gap}(W - v[|W|], d) > \frac{1}{d}(DS(\mathsf{post}_v(|W|),d) + \frac{\mu \cdot d}{d-1})$.

Next, we prove that if a finite weight sequence W is such that, the W is a bad prefix. Let Y be an arbitrary infinite but bounded weight sequence. Then $DS(W \cdot Y, d) = DS(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) + \frac{1}{d^{|W|-1}} \cdot (\mathsf{gap}(v[|W|],d) - \mathsf{gap}(v[|W|],d))$. By re-arrangement of terms we get that $DS(W \cdot Y, d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W - v[|W|], d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) + \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(v[|W|], d)$. Since $\mathsf{gap}(W - v[|W|], d) > \frac{1}{d} \cdot (DS(\mathsf{post}_v(|W|), d) + \frac{\mu \cdot d}{d-1})$ holds,

we get that $DS(W \cdot Y, d) > \frac{1}{d^{|W|}} \cdot (DS(\mathsf{post}_v(|W|), d) + \frac{\mu \cdot d}{d-1}) + \frac{1}{d^{|W|}} \cdot DS(Y, d) + \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(v[|W|], d)$. Since minimal value of $DS(Y, d) is \frac{-\mu \cdot d}{d-1}$, the inequality simplifies to $DS(W \cdot Y, d) > \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(v[|W|], d) + \frac{1}{d^{|W|}} \cdot DS(\mathsf{post}_v(|W|), d) \implies DS(W \cdot Y, d) > DS(v, d) = v$. Therefore, W is a bad prefix. ■

Intuitively, Lemma 8.6 says that if an infinite length weight sequence A has a finite prefix for which its recoverable gap is too large, then the sequence A will not be present in the coveted language. This way, if we track the recoverable gap value of finite-prefixes of A, Lemma 8.6 gives a handle for when to reject A from the coveted language. It also says that if the recoverable gap of a finite-prefix exceeds the bounds in the lemma statement, then there is no need to track the recoverable gap of other finite-prefixes any further.

Similarly, we define *very-good prefixes* for the comparison language with μ, d, $\leq$ and v as follows: A finite and bounded weight-sequence W is a *very good* prefix for the aformentioned language if for all infinite, bounded extensions of W by Y, $DS(W \cdot Y, d) \leq v$. A proof similar to Lemma 8.6 proves an upper bound for the recoverable gap of very-good prefixes of the language:

Lemma 8.7 *Let $\mu > 0$ be the integer upper bound, $d > 1$ be an integer discount factor, and the relation* inc *be the inequality $\leq$. Let $v \in \mathbb{Q}$ be the threshold value such that $v = v[0]v[1] \ldots v[m](v[m+1]v[m+2] \ldots v[n])^{\omega}$. Let W be a non-empty, bounded, finite weight-sequence. Weight sequence W is a very good-prefix of DS comparison language with μ, d, $\leq$ and v iff* $\mathsf{gap}(W - v[|W|], d) \leq \frac{1}{d} \cdot DS(\mathsf{post}_v(|W|), d) - \frac{\mu}{d-1}$.

Proof 83 Let W be a very good prefix. Then for all infinite, bounded sequences Y, we get that $DS(W \cdot Y, d) \leq v \implies DS(W, d) + \frac{1}{d^{|W|}} \cdot DS(Y, d) \leq v$. By re-arrangement of terms, we get that $\mathsf{gap}(W - v[|W|], d) \leq \frac{1}{d} \cdot DS(\mathsf{post}_v(|W|), d) - \frac{1}{d} \cdot$

$DS(Y,d)$. Since maximal value of $DS(Y,d) = \frac{\mu \cdot d}{d-1}$, we get that $\mathsf{gap}(W - v[|W|], d) \leq \frac{1}{d} \cdot DS(\mathsf{post}_v(|W|), d) - \frac{\mu}{d-1}$.

Next, we prove the converse. We know $DS(W \cdot Y, d) = DS(W, d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W,d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) + \frac{1}{d^{|W|-1}} \cdot (\mathsf{gap}(v[|W|],d) - \mathsf{gap}(v[|W|],d))$. By re-arrangement of terms we get that

$DS(W \cdot Y, d) = \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(W - v[|W|], d) + \frac{1}{d^{|W|}} \cdot DS(Y,d) + \frac{1}{d^{|W|-1}} \cdot \mathsf{gap}(v[|W|], d)$. From assumption we derive that $DS(W \cdot Y, d) \leq \frac{1}{d^{|W|}} \cdot DS(\mathsf{post}_v(|W|), d) - \frac{\mu}{d-1} + \frac{1}{d^{|W|}} \cdot DS(Y,d) + DS(v[|W|], d)$. Since maximal value of $DS(Y,d)$ is $\frac{\mu}{d-1}$, we get that $DS(W \cdot Y, d) \leq v$. Therefore, W is a very good prefix. ■

Intuitively, Lemma 8.7 says that if an infinite-length weight sequence A has a finite-prefix for which the recoverable gap is too small, A is present in the coveted language. This condition gives a handle on when to accept the weight sequence A by tracking on the recoverable gap of its finite prefixes. It also says that if the recoverable gap of a finite-prefix falls below the bound in Lemma 8.7, there is no need to track recoverable gaps of finite-prefixes of A any further.

The question we need to answer to obtain the targeted Büchi automata is how to track the recoverable gaps of finite-prefixes using a finite amount of memory (or states). We already know that there are upper and lower bounds on which recoverable gaps are *interesting*: If a recoverable gap is too large (Lemma 8.6) or too small (Lemma 8.7), the recoverable gap value does not need to be tracked any more. Now note that since the discount factor is an integer, in our case the recoverable gap value is always an integer (from definition of recoverable gap). Therefore, between the upper and lower bounds for recoverable established by Lemma 8.6 and Lemma 8.7, there are only finitely many candidate values of recoverable gaps. These finitely many values will form the states of the automata.

The final question is how to determine the transition function of the automata. For this, observe that the recoverable-gap has an inductive definition i.e. $\mathsf{gap}(\varepsilon, d) = 0$, where ε is the empty weight-sequence, and $\mathsf{gap}(W \cdot w, d) = d \cdot \mathsf{gap}(W, d) + w$, where $w \in \{-\mu, \ldots, \mu\}$. Therefore, transitions between states are established by mimicking the inductive definition.

Therefore, the formal construction of the Büchi automata for comparison language with threshold v with relation $\leq$ is given as follows:

Theorem 8.6 *Let $\mu > 0$ be the integer upper bound, $d > 1$ be an integer discount factor, and the relation* inc *be the inequality $\leq$. Let $v \in \mathbb{Q}$ be the threshold value such that $v = v[0]v[1]\ldots v[m](v[m+1]v[m+2]\ldots v[n])^{\omega}$. Then the DS comparison language for with μ, d, $\leq$ and v is ω-regular.*

Proof 84 The ideas described above are formalized to construct the desired Büchi automaton as follows:

For $i \in \{0, \ldots, n\}$, let $\mathsf{U}_i = \frac{1}{d} \cdot DS(\mathsf{post}_v(i), d) + \frac{\mu}{d-1}$ (from Lemma 8.6).

For $i \in \{0, \ldots, n\}$, let $\mathsf{L}_i = \frac{1}{d} \cdot DS(\mathsf{post}_v(i), d) - \frac{\mu}{d-1}$ (from Lemma 8.7).

The Büchi automata $\mathcal{A} = (S, s_I, \Sigma, \delta, \mathcal{F})$ is defined as follows:

- States $S = \bigcup_{i=0}^{n} S_i \cup \{\mathsf{bad}, \mathsf{veryGood}\}$ where $S_i = \{(s, i) | s \in \{\lfloor \mathsf{L}_i \rfloor + 1, \ldots, \lfloor \mathsf{U}_i \rfloor\}\}$
- Initial state $s_I = (0, 0)$, Accepting states $\mathcal{F} = S \setminus \{\mathsf{bad}\}$
- Alphabet $\Sigma = \{-\mu, -\mu + 1, \ldots, \mu - 1, \mu\}$
- Transition function $\delta \subseteq S \times \Sigma \to S$ where $(s, a, t) \in \delta$ then:
  1. If $s \in \{\mathsf{bad}, \mathsf{veryGood}\}$, then $t = s$ for all $a \in \Sigma$
  2. If s is of the form (p, i), and $a \in \Sigma$

(a) If $d \cdot p + a - v[i] > \lfloor \mathsf{U}_i \rfloor$, then $t = \mathsf{bad}$

(b) If $d \cdot p + a - v[i] \leq \lfloor \mathsf{L}_i \rfloor$, then $t = \mathsf{veryGood}$

(c) If $\lfloor \mathsf{L}_i \rfloor < d \cdot p + a - v[i] \leq \lfloor \mathsf{U}_i \rfloor$,

i. If $i == n$, then $t = (d \cdot p + a - v[i], m + 1)$

ii. Else, $t = (d \cdot p + a - v[i], i + 1)$

■

Corollary 8.3 *Let $\mu > 0$ be the integer upper bound, $d > 1$ be an integer discount factor, and* inc *be the equality or inequality relation. Let $v \in \mathbb{Q}$ be the threshold value such that $v = v[0]v[1] \dots v[m](v[m+1]v[m+2] \dots v[n])^\omega$. Then the DS comparator automata with with μ, d, $\leq$ and v is a safety or co-safety automata with $\mathcal{O}(\frac{\mu \cdot n}{d-1})$ states.*

Note that the (deterministic) Büchi automaton constructed in Theorem 8.6 is a safety automaton [63]. Safety/Co-safety automata for all other relations can be constructed by simple modifications to the one constructed above. Further, note that the number of states depends on the value v (number n). Lastly, this construction is tight, since prior work shows that when $v = 0$, the automaton constructed above is the minimal automata for its language [28].

8.3.2 Reduction of satisficing to solving safety or reachability games

We arrive at the core result of this section. We show that when the discount factor is an integer, satisficing is reduced to solving a safety or reachability game. This method is linear in size of the game graph, as opposed to the higher polynomial solutions of optimization via VI. Hence, satisficing is more efficient and scalable alternative to analyze quantitative graph games for integer discount factors.

The key idea behind this reduction is as follows: Recall, the satisficing problem is to determine whether player P_1 has a strategy that can guarantee that all resulting plays will have cost less than (or less than or equal to) a given threshold value $v \in \mathbb{Q}$. When the discount factor is an integer, the criteria for satisficing to hold is the same as ensuring that every resulting play is accepted by the safety/co-safety comparator automata with the same discount factor inequality relation, and threshold value. This lets us show that when the discount factor is an integer, the satisficing problem reduces to solving a *new game* obtained by taking the product of the quantitative graph game with the appropriate comparator. Finally, the resulting *new game* will be safety or reachability depending on whether the comparator is safety or co-safety, respectively.

The formal reduction is as follows: Let $G = (V = V_0 \uplus V_1, v_{init}, E, \gamma)$ be a quantitative graph game. Let the maximum weight on the graph game be the integer $\mu > 0$. Let $d > 1$ be the integer discount factor. Suppose, $v = v_0 v_1 \cdot v_m (v_{m+1} \cdots v_n)^\omega$ is the rational threshold value for the satisficing problem, and suppose the inequality in the problem is $\mathsf{inc} \in \{<, \leq\}$.

Then, first construct the ω-regular comparator automaton for μ, d, inc and v. Let us denote it by $\mathcal{A} = (S, s_I, \Sigma, \delta, \mathcal{F})$. From Corollary 8.3 we know that $\mathcal{A}$ is a deterministic safety or co-safety automaton. Next, construct structure $\mathsf{GA} = (W = W_0 \cup W_1, s_0 \times \mathsf{init}, \delta_W, \mathcal{F}_W)$ from G and $\mathcal{A}$ as follows:

- $W = V \times S$. Specifically $W_0 = V_0 \times S$ and $W_1 = V_1 \times S$.

  Since V_0 and V_1 are disjoint, W_0 and W_1 are disjoint too. Let sets of states W_0 and W_1 belong to players P_0 and P_1 in GA.

- Let $s_0 \times \mathsf{init}$ be the initial state of GA.

- Transition relation $\delta_W = W \times W$ is defined such that transition $(w, w') \in \delta_W$ where $w = (v, s)$ and $w' = (v', s')$ if
  - transition $(v, v') \in \delta$,
  - $n = \gamma((v, v'))$, i.e., n is the cost of the transition in G, and
  - $(s, n, s') \in \delta_C$ is a transition in the comparator automata.
- $\mathcal{F}_W = V \times \mathcal{F}$

Finally, let GA be a reachability game if the comparator $\mathcal{A}$ is a co-safety automaton, and let GA be a safety game otherwise. Correspondingly, $\mathcal{F}_W$ will be referred to as accepting states or rejecting states.

Let us call a play in the quantitative graph game G to be *winning play for* P_1 if its cost relates to threshold v by inc. Recall, the definition of winning plays in reachability and safety games (Chapter 2). Then, it is easy to show the following correspondence by relating plays in $\mathcal{G}$ and those in GA, since $\mathcal{A}$ is deterministic:

Lemma 8.8 *There is a one-one correspondence between plays in G and plays in GA. This correspondence preserves the winning condition.*

Proof 85 (One-one correspondence) Let $\rho = v_0 v_1 \dots$ be a play in G with cost sequence $w_0 w_1 \dots$. Let $s_0 s_1 \dots$ be the run of weight cost of $w_0 w_1 \dots$ in the DS comparator. Note, that since the comparator is deterministic, there is a unique run for each finite/infinite cost sequence. Then, play $\rho = v_1 v_2 \dots$ in G and corresponds to play $(v_0, s_0)(v_1, s_1) \dots$ in GA. Similarly, one can prove that for each play $(v_0, s_0)(v_1, s_1) \dots$ in GA uniquely corresponds to play $v_0 v_1 \dots$ in G, where the correspondence is defined by the weight sequence shared by the run $s_0 s_1 \dots$ in the comparator and play $v_0 v_1 \dots$ in G.

(Winning-condition preservation) A play $\rho = v_0 v_1 v_2 \ldots$ is a winning run for player P_1 in G iff its cost relates to threshold value v by inc. So, let $n_1 n_2 \ldots$ be the cost sequence of ρ in G, then $DS(n_1 n_2 \ldots, d)$ inc v holds. Let the run of weight sequence $n_0 n_1 \ldots$ be $s_0 s_1 s_2 \ldots$ in comparator automata $\mathcal{A}$. Then, run $s_0 s_1 s_2 \ldots$ is an accepting run in $\mathcal{A}$. As a result, the corresponding play in GA, i.e. $s_0 s_1 s_2 \ldots$ must be a winning run in G.

■

Next, let us call a strategy of player P_1 a *winning strategy for* P_1 in G if the strategy guarantees that all resulting plays in GA are winning for P_1. Then, it is easy to see that the winning play preserving, one-one correspondence between plays in GA and G obtained in Lemma 8.8 can be lifted to winning strategies between both games. As a result, we get the following:

Lemma 8.9 *Let G be a quantitative graph game, and $d > 1$ be an integer discount factor. Then player P_1 has a winning strategy for the satisficing problem with threshold $v \in Q$ and relation $\mathsf{inc} \in \{<, \leq\}$ iff player P_1 has a winning strategy in the reachability or safety game GA constructed as above.*

Proof 86 The proof lifts Lemma 8.8 from plays to strategies. The same idea of winning runs-preserving one-one correspondence is lifted to complete this proof. Due to the similarities, we skip the proof here. ■

Then, the final statement for our reduction is:

Theorem 8.7 *Let $G = (V, v_{init}, E, \gamma)$ be a quantitative graph game. Let $\mu > 0$ be the maximum of the absolute value of costs on transitions in G. Let $d > 1$ be an integer discount factor $v = v[0]v[1] \cdots v[m](v[m+1] \cdots v[n])^{\omega}$ be the rational threshold value,*

and $\mathsf{inc} \in \{\leq, <\}$ *be the inequality relation. Then, the complexity of satisficing on* G *with threshold* v *and inequality* inc *is* $\mathcal{O}((V + E) \cdot \mu \cdot n)$.

Proof 87 From Lemma 8.9 we know satisficing G is equivalent to solving the reachability/safety game GA In order to determine its complexity of solving GA, we need to know its number of states and edges. Clearly, there are $\mathcal{O}(|V| \cdot \mu \cdot n)$ states in GA, since the comparator $\mathcal{A}$ has $\mathcal{O}(\mu \cdot n)$ states. We claim that GA has $\mathcal{O}(|E| \cdot \mu \cdot v)$ edges. For this, we observe that a state (v, s) in GA has the same number of outgoing edges as state v in G, as comparator $\mathcal{A}$ is deterministic. Since there are $\mathcal{O}(\mu \cdot n)$ copies of each state v in GA, there are $\mathcal{O}(|E| \cdot \mu \cdot n)$ edges in GA. Therefore, the solving the reachability/safety game is $\mathcal{O}((|V| + |E|) \cdot \mu \cdot n)$. ■

Therefore, we observe that when the discount factor is an integer, our comparator-based solution for satisficing is more efficient than optimization via VI by degrees of magnitude in size of the game graph.

8.4 Satisficing via value iteration

So far, we have observed that for integer discount factors, comparator-based satisficing is more efficient than VI-based optimization. However, a crucial question still remains unanswered: Does the complexity improvement between VI for optimization to comparators for satisficing arise from (a) solving the decision problem of satisficing instead of optimization, or (b) adopting the comparator approach instead of numerical methods? To answer this, we design and analyze a VI based algorithm for satisficing: If this algorithm shows improvement over VI for optimization, then the complexity gain would have occurred from solving satisficing. Otherwise, if this algorithm does not reflect any improvement, then the gain should have come from

adhering to comparator based methods.

The VI based algorithm for satisficing is described as follows: Perform VI as was done for optimization in §3.1. Terminate the algorithm after whichever of the following two occurs first: (a) VI has performed the number of iterations as shown in Theorem 8.1, or (b): In the k-th iteration, the threshold value v falls outside of the interval defined in Lemma 8.1. In either case, one can determine how the threshold value relates to the optimal value, and hence determines satisficing.

Clearly, termination by condition (a). results in the same number of iterations as optimization itself (Theorem 8.1). We show that condition (b) does not reduce the number of iterations either. On the contrary, it introduces *non-robustness* to the algorithm's performance (See § 8.1). The reason is that the number of iterations is based on the distance between the threshold value and the optimal value. So, if this distance is large, few iterations will be required since the interval length in Lemma 8.1 declines exponentially. But if the distance is small (say close to 0), then it will take as many iterations as taken by optimization itself. Formally,

Theorem 8.8 *Let $d > 1$ be an integer discount factor. Let G be a quantitative graph game with state set V. Let W be the optimal cost, and $v \in \mathbb{Q}$ be the threshold value. Then number of iterations taken by VI for satisfaction is* $\min\{O(|V|^2), \log \frac{\mu}{|W|-v}\}$.

Hence, VI for satisficing does not improve upon VI for optimization. This indicates comparator-based methods may be responsible for the improvement.

8.5 Implementation and Empirical Evaluation

The goal of the empirical analysis is to determine whether the practical performance of these algorithms resonate with our theoretical discoveries.

Implementation details

We implement three algorithms: (a) VIOptimal: Optimization tool based on the value-iteration, (b). CompSatisfice: Satisfying tool implementing our comparator-based method, and (c)VISatisfice: Satisficing tool based on value iteration. All tools have been implemented in C++. To overcome floating-point errors in VIOptimal and VISatisfice, the tools invoke open-source arbitrary precision arithmetic library GMP (GNU Multi-Precision) [3]. Arithmetic operations in CompSatisfice are contained within integers only. Therefore, CompSatisfice does not call the GMP library.

Design and setup for experiments

Since empirical evaluations and applications of synthesis with quantitative constraints is an emerging area of research, there exist too few benchmarks of quantitative game graphs to generate a substantial amount of data to make inferences on algorithm performance. To this end, our benchmarks are derived from specification used in synthesis from the temporal specifications. We obtain (non-quantitative) graph games by converting temporal specifications to their automaton/game form. Then, we randomly assign an integer weight between $-\mu$ and μ (for a given $\mu > 0$) over all transitions. In this way, the benchmarks we create retain the structural aspects of a game graph, and hence are a better fit than randomly generated graph games.

In all, we create 291 benchmarks. We use temporal specifications used in prior literature in synthesis from temporal specifications [36,92]. The state-of-the-art logic-to-automaton conversion tool Lisa [27] has been deployed to obtain the automaton/-graph game. The number of states in our benchmark set ranges from 3 to 50000+. Discount factor $d = 2$, threshold v ranges in 0-10. All experiments were run on 8 CPU cores at 2.4GHz, 16GB RAM on a 64-bit Linux machine.

Observations and Inferences

CompSatisfice **outperforms** VIOptimal [*] in runtime and consequently in number of benchmarks solved The cactus plot in Fig 8.2 clearly indicates that CompSatisfice is more efficient than VIOptimal. In fact, a closer look reveals that all benchmarks solved by VIOptimal have fewer than 200 states. In contrast, CompSatisfice solves all but two benchmarks. Thus solving benchmarks with many thousands of states.

To test scalability, we plot runtime of both tools on a set of scalable benchmarks. For integer parameter $i > 0$, the i-th scalable benchmark has $3 \cdot 2^i$ states. Fig 8.3 plots the observations in log-log scale. Thus, the slope of the straight line indicates the degree of polynomial (in practice). We see that in practice CompSatisfice exhibits linear behavior (slope $\sim$1), whereas VIOptimal is much more expensive (slope $>> 1$) even for small values of weights and threshold.

CompSatisfice **is more robust than** VISatisfice**.** We compare CompSatisfice and VISatisfice as the threshold value changes. This experiment is chosen due to Theorem 8.8 which proves that VISatisfice is non-robust. The goal is to see how VISatisfice fluctuates in practice, while CompSatisfice maintains steady performance owning to its low complexity. These are shown in Fig 8.4.

8.6 Quantitative games with temporally extended goals

In several synthesis tasks, the objective is to solve a quantitative game while also satisfying a given temporal goal. So, the goal is to ensure that the system has a strategy that also ensures that the temporal objective is met along all possible plays

[*]Figures are best viewed online and in color

resulting from the game. However, prior works have proven that optimal strategies may not exist under extension with temporal goals, rendering analysis by optimization incompatible with temporal goals [41]. In this section, we show that our comparator-based solution for satisficing can extend to all kinds of temporal goals.

Intuitively, a *quantitative game with temporal goals* appends a quantitative game with a *labeling function* that assigns states to atomic propositions. Therefore, each play of the game is associated with the sequence of atomic propositions. It is over this sequence of atomic propositions that the given temporal goal is evaluated. In this section, we show that our comparator-based approach to solve satisficing on quantitative games can be extended to all temporal goals expressed by *linear temporal logic (LTL)* [77]. More generally, our approach extends to all ω-regular goals. This is in sharp contrast to existing work based on VI which extends to a limited subclass of temporal goals only, namely safety objectives [91], since optimal solutions are not known to exist with ω-regular objectives [41].

Formally, a quantitative game with temporal goals, denoted by $\mathcal{G}_{\mathsf{T}}$, is the tuple $(G, AP, \mathcal{L}, \varphi)$ where G is a quantitative game, AP is a finite set of atomic propositions, $\mathcal{L} : V \to 2^{AP}$ is a labeling function, and φ is an LTL formula over propositions AP. Plays, cost-sequences, cost and strategies are defined as earlier for quantitative games. The difference is that plays in a quantitative game with temporal goals are also accompanied with a sequence of propositions. If $\rho = v_0 v_1 v_2 \dots$ is a play in game $\mathcal{G}_{\mathsf{T}}$, then $\rho_{AP} = A_0 A_1 A_2 \dots$ is the *proposition sequence* where $A_i = \mathcal{L}(v_i) \subseteq AP$ for all $i \geq 0$. Given an LTL formula φ over AP, we say that a play ρ satisfies φ if its corresponding proposition sequence ρ_{AP} satisfies φ.

Let us assume that the objectives of P_0 and P_1 are to maximize and minimize the cost of plays, respectively. Given threshold value $v \in \mathbb{Q}$ and inequality relation

$\mathsf{inc} \in \{\leq, <\}$, a strategy for P_1 is said to be *winning* in a quantitative game with temporal goals $\mathcal{G}_\mathsf{T}$, threshold v and relation inc if the strategy is winning for P_1 in quantitative game G with threshold v and relation inc, and every resulting play in the game satisfies φ.

The satisficing problem with temporal goals is defined as follows:

Definition 8.4 (Satisficing problem with temporal goals) Given a quantitative game with temporal goals $\mathcal{G}_\mathsf{T} = (G, AP, \mathcal{L}, \varphi)$, a threshold value $v \in \mathbb{Q}$, and an inequality relation $\mathsf{R} \in \{\leq, <\}$, the *satisficing problem with temporal goals* is to determine whether P_1 has a winning strategy in $\mathcal{G}_\mathsf{T}$ with threshold v and relation inc, assuming that the objectives of P_0 and P_1 are to maximize and minimize the cost of plays, respectively.

We show that solving the satisficing problem with temporal goals reduces to solving a parity game. The key insight behind this is that when the discount factor is an integer then both the satisficing criteria and the temporal goal are represented by NBAs. As a result of which the criteria for the player to win a satisficing problem with temporal goals can be represented by a single NBA that refers to the combination of both conditions. Thus, leading to a parity game.

Theorem 8.9 *Let $\mathcal{G}_\mathsf{T} = (G, AP, \mathcal{L}, \varphi)$ be a quantitative game with temporal goals. Let V and E refer to the set of states and edges in game G, and μ be the maximum of the absolute value of weights along transitions in G. Let $d > 1$ be an integer discount factor, and $v = v[0]v[1]\cdots v[m](v[m+1]\cdots v[n])^\omega$ be the rational threshold value*

- *Solving the satisficing problem with temporal goals reduces to solving a parity game. The size of the parity game is linear in $|V|$, μ, n and double exponential in $|\varphi|$.*

- *If φ can be represented by a (deterministic) safety/co-safety automata $\mathcal{A}$, then solving the satisficing problem with temporal goals is $\mathcal{O}((|V|+|E|)\cdot\mu\cdot n\cdot|\mathcal{A}|)$, where $|\mathcal{A}| = 2^{2^{\mathcal{O}(|\varphi|)}}$.*

Proof 88 This reduction can be carried out in two steps. In the first step, we reduce the quantitative graph game G to a reachability/safety game GA by means of the comparator automata for DS with integer discount factor, as done in Section 8.3.2. There is one key difference in constructing GA. The labelling function of $\mathcal{G}_\mathsf{T}$ is extended to states in GA so that the label of state (v, s) in GA is identical to the label of state v in $\mathcal{G}_\mathsf{T}$. In the second step, we incorporate the temporal goal into the safety/reachability game GA. For this, first the temporal goal φ is converted into its equivalent *deterministic parity automaton (DPA)* [87]. Next, take the product of the game GA with the DFA by synchronizing the labeling function of GA with transitions in the DPA. This follows standard operations for product constructions. The end result will be a parity game that is linear in the size of GA and the DPA for φ. Recall from Section 8.3.2, size of GA is linear in size of G, μ and n. Further, note that the DPA is double exponential in size of φ. Therefore, the final product game has size linear in $|V|$ and μ, and double exponential in $|\varphi|$.

Finally, in the special case where the DPA for φ is actually a deterministic safety/co-safety automaton, the resulting product game can be solved in time linear in the size of the final product game. The reason behind this is that the final DPA will be a weak Büchi automata generated from the union of safety/co-safety automaton from the comparator, and a safety/co-safety autoamton from the temporal property (Theorem 6.1). Games with weak Büchi winning conditions can be solved in size linear to the underlying graph game.

Finally, the proof of correctness is identical to that of Lemma 8.9. ■

8.7 Chapter summary

This work introduced the notion of satisficing for quantitative games with discounted-sum aggregation function. This is proposed as a decision variant of the optimization problem. Following a thorough analysis of both problems, we show that satisficing theoretical, empirical and practical advantages over the optimization problem. In particular, we show that when the discount factor is an integer, then the satisficing problem can be solved by automata-based solutions. Not only is our automata-based solution for satisficing more efficient than existing solutions for optimization in size of the game graph, our solutions perform more scalablely and robustly in empirical evaluations. Furthermore, the practical applicability of quantitative games with temporal goals can be handled naturally with our automata-based solution for satisficing as opposed to the non-existence of optimal solutions with temporal goals. This works yet again presents the benefits of automata-based approaches in quantitative reasoning over traditional numerical approaches.

While this chapter explored the automata-based solution for an integer discount factor only, in theory these results could be extended to non-integer discount factors as well as done in Chapter 7. This may come at the cost of a small approximation factor, but we can envision solving *approximate* satisficing using the approximate comparators for non-integer discount factors. Of course, the critical question here is to investigate the impact of the approximation in practical usage.

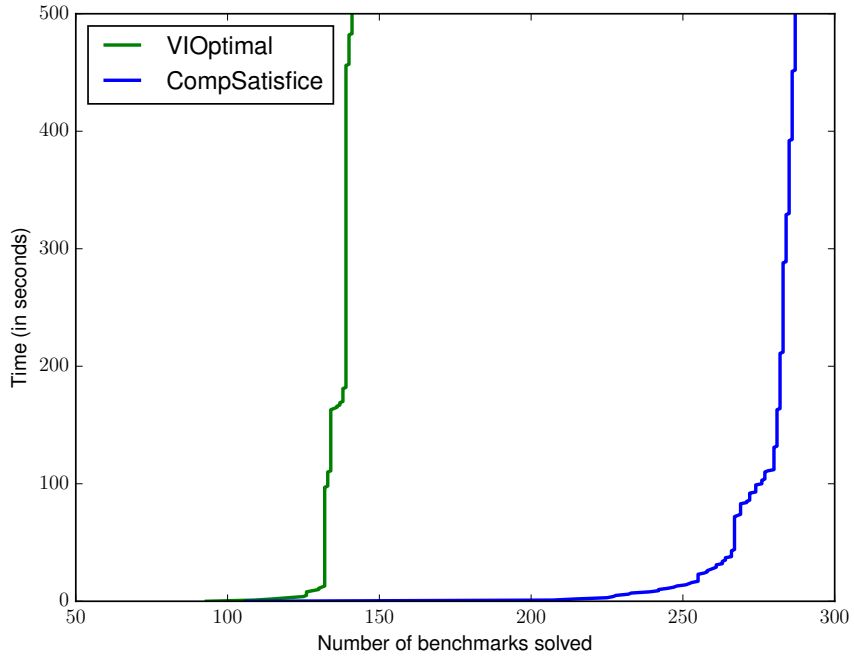


Figure 8.2 : Cactus plot. $\mu = 5, v = 3$. Total benchmarks = 291

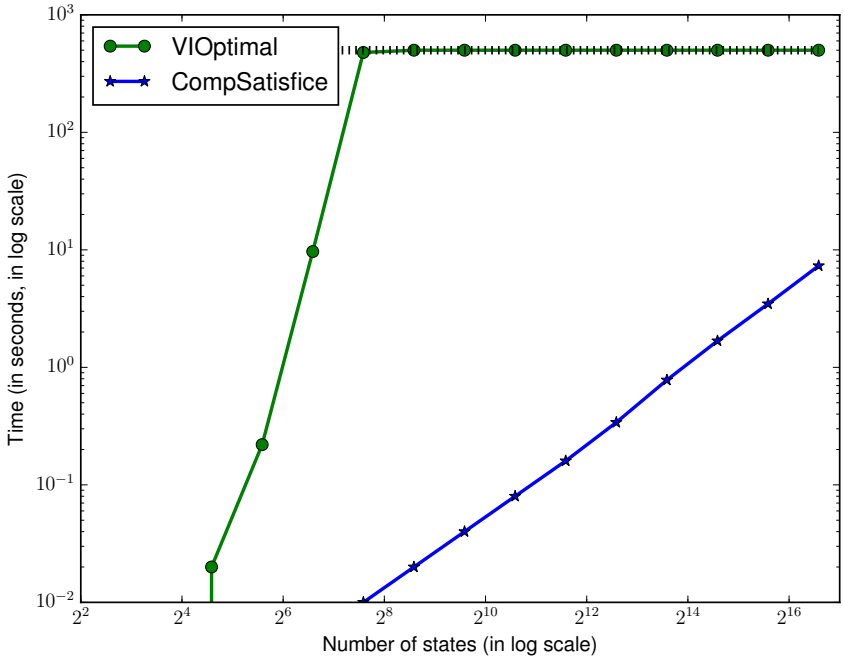


Figure 8.3 : Single counter scalable benchmark. $\mu = 5, v = 3$. Timeout = 500s.

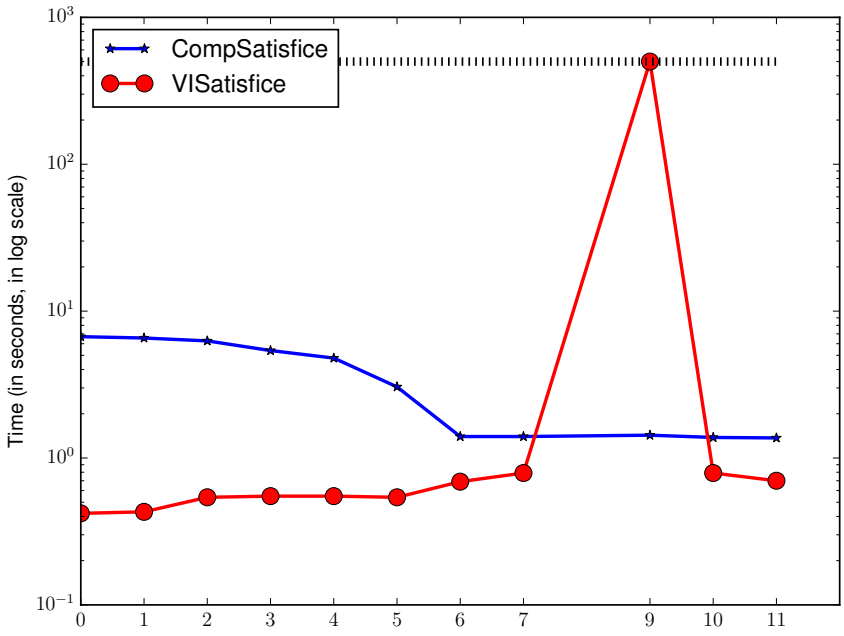


Figure 8.4 : Robustness. Fix benchmark, vary v. $\mu = 5$. Timeout = 500s.

Chapter 9

Conclusion

9.1 Concluding remarks

This thesis introduces *comparator automata*, a novel technique based on automata for quantitative reasoning. The use of automata for quantitative reasoning is unique in its own, and infact counterintuitive as far as earlier work is concerned.

The challenges of lack of genralizability and separation-of-techniques adversely affect the practical viability of quantitative reasoning, and also prevent a clear understanding of similarities and dissimilarities across aggregate functions. The motivation behind the development of comparator automata is to addresses these challenges. Our investigations on quantitative inclusion and quantitative games, especially an in-depth examination of the discounted-sum aggregation function, have established key theoretical results, improved upon the state-of-the-art in empirical performance, and indicated promise of approach in applications.

An undercurrent in the progress of comparator-based approaches is that they have bridged quantitative reasoning to qualitative reasoning. This is in stark contrast to separation-of-techniques in prior work where the two are segregated from one another. As demonstrated in this thesis, this digression will prove to be of immense importance in the long run, since it allows quantitative reasoning to leverage the advances in the theory, practice and applications of qualitative reasoning. To the best of our knowledge, this thesis is the first work to do so.

9.2 Future work

Theory: Building on the foundations.

ω-regular comparators advocate for the treatment of quantitative properties as any other ω-regular property. As a result, it is likely that problems that have found success with an ω-regular property, such as a temporal logic [77], could do so with ω-regular comparators as well. The study of quantitative games with imperfect information gives a glimpse of the hypothesis. One avenue of interest are probabilistic systems. So far, probabilistic verification and synthesis involves techniques from linear programming, optimization, solving sets of equalities and inequalities, and so on. Whether comparator-based approaches can substitute some of these operations is an open question.

Another line of research is to identify necessary and sufficient conditions for aggregate functions to have ω-regular comparators. So far, aggregate functions have been studied on a case-by-case basis. The broadest classification we have currently is that it is sufficient if an aggregate function is ω-regular. A better understanding of necessary and sufficient conditions could contribute to clearer understanding of aggregate functions.

Practice: Scaling further and beyond.

Symbolic methods have been gaining in popularity in qualitative reasoning due to their scalability and efficiency advantages over explicit methods [35]. In symbol methods, the state space is typically represented with a logarithmic number of bits. Following this success, the symbolic reasoning has been adopted in quantitative reasoning. In most cases, however, the symbolic structures extend their qualitative counterparts.

For example, Binary Decision Diagram (BDDs) [34] are a popular symbolic data-structure in qualitative domains. These have been extended to MTBDDs and ADDs to reason about quantitative properties. Since the development of quantitative symbolic data-structures and their algorithms are still in nascent stages, relative to their qualitative counterparts, their full impact is yet to be discovered.

While investigations on quantitative symbolic methods are underway, comparators offer an alternate perspective. Since comparators are automata, existing symbolic methods such as BDD, SAT and SMT can be applied. Albeit promising, only a comparative analysis can reveal the strengths and weakness of quantitative and comparator-based qualitative symbolic methods, and expose areas for improvement.

Application: White-boxing RL engines.

Owing to the unprecedented rise of machine learning algorithms, reinforcement learning (RL) has come to the forefront in the design of controllers for robotics, autonomous vehicles, and several control-related domestic appliances. Due to their safety-critical nature of these applications, it has become crucial to *white-box* the underlying RL algorithms. In response, the formal methods community has begun looking into the quantitative and qualitative aspects of RL algorithms.

This direction is particularly exciting for comparator automata. First of all, several RL engines make use of discounted-sum. Secondly, with comparators we can combine quantitative and qualitative properties with rigorous guarantees, as demonstrated in this thesis. Having said that, the capabilities of comparators will have to be extended further to be applicable to the domain. For example, RL algorithms have to work under the assumption that the reward function is known partially only. To address this, comparators will have to be effective at verification and synthesis under

partial information. Similarly, one could identify numerous ways in which automata-theoretic reasoning could be of use in the space of white-boxing RL engines.

In all, this thesis has introduced a novel framework for the formal analysis of quantitative systems. The evidence collected throughout the thesis indicates promise of our approach. The scope in future directions of research spreads across the wide spectrum from theoretical results to current applications. In all we believe that this thesis has managed to work at the tip of automata-based reasoning of quantitative systems. The iceberg is yet to be discovered.

Appendix A

Appendix of miscellaneous results

A.1 Discounted-sum is an ω-regular function

We use the ω-regular comparator for DS-aggregate function for integer discount-factor to prove that discounted-sum with integer discount-factors is an ω-regular aggregate function.

Theorem A.1 *Let $d > 1$ be a non-integer, rational discount-factor. The discounted-sum aggregate function with discount factor d is not ω-regular.*

Proof 89 Immediate from Lemma 4.1 and Theorem 4.9.

Theorem A.2 *Let $d > 1$ be an integer discount-factor. The discounted-sum aggregate function with discount-factor d is ω-regular under base d.*

Proof 90 We define the discounted-sum aggregate function automaton (DS-function automaton, in short): For integer $\mu > 0$, let $\Sigma = \{0, 1, \ldots \mu\}$ be the input alphabet of DS-function, and $d > 1$ be its integer base. Büchi automaton $\mathcal{A}_d^\mu$ over alphabet $\Sigma \times \mathsf{AlphaRep}(d)$ is a DS-function automaton of type $\Sigma^\omega \to \mathbb{R}$ if for all $A \in \Sigma^\omega$, $(A, \mathsf{rep}(DS(A, d)), d) \in \mathcal{A}_d^\mu$. Here we prove that such a $\mathcal{A}_d^\mu$ exists.

Let $\mu > 0$ be the integer upper-bound. Let $\mathcal{A}_d^=$ be the DS-comparator for integer discount-factor $d > 1$ for relation $=$. Intersect $\mathcal{A}_d^=$ with the Büchi automata consisting of all infinite words from alphabet $\{0, 1 \ldots \mu\} \times \{0, \ldots, d-1\}$. The re-

sulting automaton $\mathcal{B}$ accepts (A, B) for $A \in \{0, \ldots, \mu\}^\omega$ and $B \in \{0, \ldots, d-1\}^\omega$ iff $DS(A, d) = DS(B, d)$.

Since all elements of B are bounded by $d-1$, $DS(B, d)$ can be represented as an ω-word as follows: Let $B = B[0], B[1] \ldots$, then its ω-word representation in base d is given by $+ \cdot (\mathsf{Int}(DS(B, d), d), \mathsf{Frac}(DS(B, d), d))$ where $\mathsf{Int}(DS(B, d), d) = B[0] \cdot 0^\omega$ and $\mathsf{Frac}(DS(B, d), d) = B[1], B[2] \ldots$. This transformation of integer sequence B into its ω-regular word form in base d can be achieved with a simple transducer $\mathcal{T}$.

Therefore, application of transducer $\mathcal{T}$ to Büchi automaton $\mathcal{B}$ will result in a Büchi automaton over the alphabet $\Sigma \times \mathsf{AlphaRep}(d)$ such that for all $A \in \Sigma^\omega$ the automaton accepts $(A, \mathsf{rep}(DS(A, d), d))$. This is exactly the DS-function automaton over input alphabet Σ and integer base $d > 1$. Therefore, the discounted-sum aggregate function with integer discount-factors in ω-regular. ■

Recall, this proof works only for the discounted-sum aggregate function with integer discount-factor. In general, there is no known procedure to derive a function automaton from an ω-regular comparator (Conjecture 4.1).

A.2 Connection between discounted-sum and sum

Theorem A.3 *Let integer $\mu > 0$ be the upper-bound. Let $d = 1 + 2^{-k}$ be the discount-factor for rational-number $k > 0$. Let A be a non-empty, bounded, finite, weight-sequence. Then* $\mod \mathsf{Sum}(A) - DS(A, d) < 2^{-k} \cdot \mu \cdot |A|^2$

Proof 91 We prove the desired bound on the difference between discounted-sum and

sum. Let $n > 0$ be the length of A, and $\varepsilon = 2^{-k}$.

$$
\begin{aligned}
|\mathsf{Sum}(A) - DS(A,d)| = &\quad \bmod \Sigma_{i=0}^{n-1} A_i - \frac{A_i}{d^i} \\
&\leq \Sigma_{i=0}^{n-1} \quad \bmod A_i \cdot \quad \bmod 1 - \frac{1}{d^i} \\
&\text{By Cauchy-Schwarz inequality, we get} \\
&\leq \sqrt{(\Sigma_{i=0}^{n-1} A_i^2) \cdot (\Sigma_{i=0}^{n-1} (1 - \frac{1}{d^i})^2)} \\
&\text{Since } A \text{ is bounded by } \mu \\
&\leq \sqrt{n \cdot \mu^2 \cdot (\Sigma_{i=0}^{n-1} (1 - \frac{1}{d^i})^2)} = \sqrt{n \cdot \mu^2 \cdot (\Sigma_{i=0}^{n-1} (1 - \frac{1}{(1+\varepsilon)^i})^2)} \\
&\leq \sqrt{n \cdot \mu^2 \cdot (\Sigma_{i=0}^{n-1} (1 - \frac{1}{(1+\varepsilon)^{(\frac{1}{\varepsilon}) \cdot \varepsilon \cdot i}})^2)} \\
&\text{Since } (1+x)^{\frac{1}{x}} < e^x, \text{ for all } 0 < x \leq 1 \\
&< \sqrt{n \cdot \mu^2 \cdot (\Sigma_{i=0}^{n-1} (1 - \frac{1}{e^{\varepsilon \cdot i}})^2)} \\
&\leq \sqrt{n \cdot \mu^2 \cdot (\Sigma_{i=0}^{n-1} (1 - e^{-\varepsilon \cdot i})^2)} \\
&\leq \sqrt{n \cdot \mu^2 \cdot n \cdot (1 - e^{-\varepsilon \cdot n})^2)} \text{ as } i < n \\
&\text{Since } 1 - x < e^{-x} \text{ holds for all } x > 0 \\
&< \sqrt{n \cdot \mu^2 \cdot n \cdot (\varepsilon \cdot n)^2)} = \varepsilon \cdot \mu \cdot n^2
\end{aligned}
$$

■

The bound above is simply an upper bound. It will be good to tighten the bound to $\mathcal{O}(\varepsilon \cdot \mu \cdot n)$.

Bibliography

[1] Satisficing. https://en.wikipedia.org/wiki/Satisficing.

[2] GLPK. https://www.gnu.org/software/glpk/.

[3] GMP. https://gmplib.org/.

[4] GOAL. http://goal.im.ntu.edu.tw/wiki/.

[5] Rabit-Reduce. http://www.languageinclusion.org/.

[6] P. A Abdulla, Y. Chen, L. Clemente, L. Holík, Chih-Duo Hong, R. Mayr, and T. Vojnar. Simulation subsumption in ramsey-based büchi automata universality and inclusion testing. In *Proc. of CAV*, pages 132–147. Springer, 2010.

[7] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukás Holík, Chih-Duo Hong, Richard Mayr, and Tomas Vojnar. Advanced ramsey-based büchi automata inclusion testing. In *Proc. of CONCUR*, volume 11, pages 187–202. Springer, 2011.

[8] Dilip Abreu. On the theory of infinitely repeated games with discounting. *Econometrica*, pages 383–396, 1988.

[9] Umair Z Ahmed, Sumit Gulwani, and Amey Karkare. Automatically generating problems and solutions for natural deduction. In *Proc. of IJCAI*, 2013.

[10] S. Akiyama, C. Frougny, and J. Sakarovitch. On the representation of numbers in a rational base. *Proc. of Word*, pages 47–64, 2005.

[11] S. Almagor, U. Boker, and O. Kupferman. Whatś decidable about weighted automata? In *Proc. of ATVA*, pages 482–491. Springer, 2011.

[12] Shaull Almagor, Udi Boker, and Orna Kupferman. Formalizing and reasoning about quality. In *In Proc. of ICALP*, pages 15–27. Springer, 2013.

[13] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

[14] R. Alur and K. Mamouras. An introduction to the streamqre language. *Dependable Software Systems Engineering*, 50:1, 2017.

[15] Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On omega-languages defined by mean-payoff conditions. In *Proc. of FOSSACS*, pages 333–347. Springer, 2009.

[16] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *Transactions on Algorithms*, 6(2):28, 2010.

[17] Garrett Andersen and Vincent Conitzer. Fast equilibrium computation for infinitely repeated games. In *Proc. of AAAI*, pages 53–59, 2013.

[18] D. Andersson. An improved algorithm for discounted payoff games. In *ESSLLI Student Session*, pages 91–98, 2006.

[19] C. Baier. Probabilistic model checking. In *Dependable Software Systems Engineering*, pages 1–23. 2016.

[20] Christel Baier, Luca de Alfaro, Vojtěch Forejt, and Marta Kwiatkowska. Model checking probabilistic systems. In *Handbook of Model Checking*, pages 963–999. Springer, 2018.

[21] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*. MIT press Cambridge, 2008.

[22] Teodora Baluta, Shiqi Shen, Shweta Shinde, Kuldeep S Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. *arXiv preprint arXiv:1906.10395*, 2019.

[23] S. Bansal. Algorithmic analysis of regular repeated games. Master's thesis, Rice University, 2016.

[24] S. Bansal, S. Chaudhuri, and M. Y. Vardi. Automata vs linear-programming discounted-sum inclusion. In *Proc. of CAV*, 2018.

[25] S. Bansal, S. Chaudhuri, and M. Y. Vardi. Comparator automata in quantitative verification. In *Proc. of FOSSACS*, 2018.

[26] S. Bansal, S. Chaudhuri, and M. Y. Vardi. Comparator automata in quantitative verification (full version). *CoRR*, abs/1812.06569, 2018.

[27] S. Bansal, Y. Li, L.M. Tabajara, and M. Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *Proc. of AAAI*, 2020.

[28] S. Bansal and M. Y. Vardi. Safety and co-safety comparator automata for discounted-sum inclusion. In *Proc. of CAV*, 2019.

[29] Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. of CAV*, pages 140–156. Springer, 2009.

[30] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.

[31] U. Boker and T. A. Henzinger. Exact and approximate determinization of discounted-sum automata. *LMCS*, 10(1), 2014.

[32] U. Boker, T. A. Henzinger, and J. Otop. The target discounted-sum problem. In *Proc. of LICS*, pages 750–761, 2015.

[33] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.

[34] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[35] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[36] Alberto Camacho, Jorge A Baier, Christian Muise, and Sheila A McIlraith. Finite LTL synthesis as planning. In *ICAPS*, pages 29–38. AAAI Press, 2018.

[37] A. Chakrabarti, K. Chatterjee, T.A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *Advanced Re-*

search Working Conference on Correct Hardware Design and Verification Methods, pages 50–64. Springer, 2005.

[38] K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. In *Proc. of LICS*, pages 199–208. IEEE, 2009.

[39] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *Transactions on Computational Logic*, 11(4):23, 2010.

[40] Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *Proc. of LICS*, pages 178–187. IEEE, 2005.

[41] Krishnendu Chatterjee, Thomas A Henzinger, Jan Otop, and Yaron Velner. Quantitative fair simulation games. *Information and Computation*, 254:143–166, 2017.

[42] Swarat Chaudhuri, Sriram Sankaranarayanan, and Moshe Y. Vardi. Regular real analysis. In *Proc. of LICS*, pages 509–518, 2013.

[43] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[44] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[45] Rina S Cohen and Arie Y Gold. Theory of ω-languages: Characterizations of

ω-context-free languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977.

[46] L. De Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Model checking discounted temporal properties. In *Proc. of TACAS*, pages 77–92. Springer, 2004.

[47] L. De Alfaro, M. Faella, and M Stoelinga. Linear and branching metrics for quantitative transition systems. In *In Proc. of ICALP*, pages 97–109. Springer, 2004.

[48] L. De Alfaro, T.A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *ICALP*, pages 1022–1037. Springer, 2003.

[49] T.L. Dean and M.S. Boddy. An analysis of time-dependent planning. In *Proc. of AAAI*, volume 88, pages 49–54, 1988.

[50] Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In *International Workshop on Computer Science Logic*, pages 260–274. Springer, 2010.

[51] Laurent Doyen and Jean-François Raskin. Antichain algorithms for finite automata. In *Proc. of TACAS*, pages 2–22. Springer, 2010.

[52] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer, 2009.

[53] Loris DAntoni, Roopsha Samanta, and Rishabh Singh. Qlose: Program repair with quantitative objectives. In *Proc. of CAV*, pages 383–401. Springer, 2016.

[54] Rüdiger Ehlers. Minimising deterministic büchi automata precisely using sat solving. In *International Conference on SAT*, pages 326–332. Springer, 2010.

[55] Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. In *Proc. of CONCUR*, pages 132–146. Springer, 2012.

[56] Bernd Finkbeiner, Christopher Hahn, and Hazem Torfah. Model checking quantitative hyperproperties. In *Proc. of CAV*, pages 144–163. Springer, 2018.

[57] Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *Journal of the ACM (JACM)*, 60(1):1–16, 2013.

[58] K. He, M. Lahijanian, L.E. Kavraki, and M.Y. Vardi. Reactive synthesis for finite tasks under resource constraints. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 5326–5332. IEEE, 2017.

[59] John E Hopcroft and Jeffrey D Ullman. Formal languages and their relation to automata. 1969.

[60] L. Hu, Q.and DAntoni. Syntax-guided synthesis with quantitative syntactic objectives. In *Proc. of CAV*, pages 386–403. Springer, 2018.

[61] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.

[62] Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *International Colloquium on Automata, Languages, and Programming*, pages 101–112. Springer, 1992.

[63] O. Kupferman and M. Y. Vardi. Model checking of safety properties. In *Proc. of CAV*, pages 172–183. Springer, 1999.

[64] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *Transactions on Computational Logic*, 2(3):408–429, 2001.

[65] Orna Kupfermant and Moshe Y Vardit. Synthesis with incomplete informatio. In *Advances in temporal logic*, pages 109–127. Springer, 2000.

[66] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, September 2007.

[67] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[68] Marta Kwiatkowska, Gethin Norman, and David Parker. Advances and challenges of probabilistic model checking. In *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1691–1698. IEEE, 2010.

[69] M. Lahijanian, S. Almagor, D. Fried, L.E. Kavraki, and M.Y. Vardi. This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In *AAAI*, pages 3664–3671, 2015.

[70] Christof Löding. Efficient minimization of deterministic weak ω-automata. volume 79, pages 105–109. Elsevier, 2001.

[71] R. Mayr and L. Clemente. Advanced automata minimization. *ACM SIGPLAN Notices*, 48(1):63–74, 2013.

[72] M. Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

[73] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[74] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[75] M.J. Osborne and A. Rubinstein. *A course in game theory*. MIT press, 1994.

[76] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[77] Amir Pnueli. The temporal logic of programs. In *Proc. of FOCS*, pages 46–57. IEEE, 1977.

[78] M.L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[79] Walter Rudin. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964.

[80] S. Safra. On the complexity of ω-automata. In *Proc. of FOCS*, pages 319–327. IEEE, 1988.

[81] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[82] Sanjit A Seshia, Ankush Desai, Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Sumukh Shivakumar, Marcell Vazquez-Chanlatte, and Xiangyu Yue. Formal specification for deep neural networks. In *Proc. of ATVA*, pages 20–34. Springer, 2018.

[83] Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

[84] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 15–26, 2013.

[85] R.S. Sutton and A.G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[86] D. Tabakov and M. Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. of LPAR*, pages 396–411. Springer, 2005.

[87] W. Thomas, T. Wilke, et al. *Automata, logics, and infinite games: A guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[88] M. Y Vardi. The büchi complementation saga. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 12–22. Springer, 2007.

[89] Moshe Y Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.

[90] Amol Wakankar, Paritosh K Pandya, and Raj Mohan Matteplackel. Dcsynth: Guided reactive synthesis with soft requirements. In *Working Conference on Verified Software: Theories, Tools, and Experiments*, pages 124–142. Springer, 2019.

[91] Min Wen, Rüdiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4983–4990. IEEE, 2015.

[92] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y Vardi. A symbolic approach to safety ltl synthesis. In *Haifa Verification Conference*, pages 147–162. Springer, 2017.

[93] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.