Comparator Automata in Quantitative Verification

Suguman Bansal^(✉), Swarat Chaudhuri^(✉), and Moshe Y. Vardi^(✉)

Rice University, Houston, TX 77005, USA
 {suguman,swarat}@rice.edu, vardi@cs.rice.edu

Abstract. The notion of comparison between system runs is fundamental in formal verification. This concept is implicitly present in the verification of qualitative systems, and is more pronounced in the verification of quantitative systems. In this work, we identify a novel mode of comparison in quantitative systems: the online comparison of the aggregate values of two sequences of quantitative weights. This notion is embodied by *comparator automata* (*comparators*, in short), a new class of automata that read two infinite sequences of weights synchronously and relate their aggregate values.

We show that comparators that are finite-state and accept by the Büchi condition lead to generic algorithms for a number of well-studied problems, including the quantitative inclusion and winning strategies in quantitative graph games with incomplete information, as well as related non-decision problems, such as obtaining a finite representation of all counterexamples in the quantitative inclusion problem.

We study comparators for two aggregate functions: discounted-sum and limit-average. We prove that the discounted-sum comparator is ω -regular for all integral discount factors. Not every aggregate function, however, has an ω -regular comparator. Specifically, we show that the language of sequence-pairs for which limit-average aggregates exist is neither ω -regular nor ω -context-free. Given this result, we introduce the notion of *prefix-average* as a relaxation of limit-average aggregation, and show that it admits ω -context-free comparators.

1 Introduction

Many classic questions in formal methods can be seen as involving *comparisons* between different system runs or inputs. Consider the problem of verifying if a system S satisfies a linear-time temporal property P . 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

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-319-89366-2_23) contains supplementary material, which is available to authorized users.

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 [5].

The need for such a formulation is clearer in quantitative systems, in which every run of a word is associated with a sequence of (rational-valued) weights. The weight of a run is given by *aggregate function* $f : \mathbb{Q}^\omega \rightarrow \mathbb{R}$, which returns the real-valued *aggregate value* of the run's weight sequence. The weight of a word is given by the supremum or infimum of the weight of all its runs. Common examples of aggregate functions include discounted-sum and limit-average.

In a well-studied class of problems involving quantitative systems, the objective is to check if the aggregate value of words of a system exceed a constant threshold value [14–16]. This is a natural generalization of emptiness problems in qualitative systems. Known solutions to the problem involve arithmetic reasoning via linear programming and graph algorithms such as negative-weight cycle detection, computation of maximum weight of cycles etc. [4, 18].

A more general notion of comparison relates aggregate values of two weight sequences. Such a notion arises in the *quantitative inclusion problem* for weighted automata [1], where the goal is to determine whether the weight of words in one weighted automaton is less than that in another. Here it is necessary to compare the aggregate value along runs between the two automata. Approaches based on arithmetic reasoning do not, however, generalize to solving such problems. In fact, the known solution to discounted-sum inclusion with integer discount-factor combines linear programming with a *specialized* subset-construction-based determinization step, rendering an EXPTIME algorithm [4, 6]. Yet, this approach does not match the PSPACE lower bound for discounted-sum inclusion.

In this paper, we present an automata-theoretic formulation of this form of comparison between weighted sequences. Specifically, 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. While comparisons between weight sequences happen implicitly in prior approaches to quantitative systems, comparator automata make these comparisons explicit. We show that this has many benefits, including generic algorithms for a large class of quantitative reasoning problems, as well as a direct solution to the problem of discounted-sum inclusion that also closes its complexity gap.

A *comparator for aggregate function* f is an automaton that accepts a pair (A, B) of sequences of bounded rational numbers iff $f(A) R f(B)$, where R is an inequality relation ($>$, $<$, $\geq$, $\leq$) or the equality relation. A comparator could be finite-state or (pushdown) infinite-state. This paper studies such comparators.

A comparator is ω -regular if it is finite-state and accepts by the Büchi condition. We show that ω -regular comparators lead to generic algorithms for a number of well-studied problems including the quantitative inclusion problem, and in showing existence of winning strategies in incomplete-information quantitative games. Our algorithm yields PSPACE-completeness of quantitative inclusion when the ω -regular comparator is provided. The same algorithm extends to obtaining finite-state representations of counterexample words in inclusion.

Next, we show that the discounted-sum aggregation function admits an ω -regular comparator when the discount-factor $d > 1$ is an integer. Using properties of ω -regular comparators, we conclude that the discounted-sum inclusion is PSPACE-complete, hence resolving the complexity gap. Furthermore, we prove that the discounted-sum comparator for $1 < d < 2$ cannot be ω -regular. We suspect this result extends to non-integer discount-factors as well.

Finally, we investigate the limit-average comparator. Since limit-average is only defined for sequences in which the average of prefixes converge, limit-average comparison is not well-defined. We show that even a Büchi pushdown automaton cannot separate sequences for which limit-average exists from those for which it does not. Hence, we introduce the novel notion of *prefix-average comparison* as a relaxation of limit-average comparison. We show that the prefix-average comparator admits a comparator that is ω -context-free, i.e., given by a Büchi pushdown automaton, and we discuss the utility of this characterization.

This paper is organized as follows: Preliminaries are given in Sect. 2. Comparator automata is formally defined in Sect. 3. Generic algorithms for ω -regular comparators are discussed in Sects. 3.1 and 3.2. The construction and properties of discounted-sum comparator, and limit-average and prefix-average comparator are given in Sects. 4 and 5, respectively. We conclude with future directions in Sect. 6.

Related Work. The notion of comparison has been widely studied in quantitative settings. Here we mention only a few of them. Such aggregate-function based notions appear in weighted automata [1, 17], quantitative games including mean-payoff and energy games [16], discounted-payoff games [3, 4], in systems regulating cost, memory consumption, power consumption, verification of quantitative temporal properties [14, 15], and others. Common solution approaches include graph algorithms such as weight of cycles or presence of cycle [18], linear-programming-based approaches, fixed-point-based approaches [8], and the like. The choice of approach for a problem typically depends on the underlying aggregate function. In contrast, in this work we present an automata-theoretic approach that unifies solution approaches to problems on different aggregate functions. We identify a class of aggregate functions, ones that have an ω -regular comparator, and present generic algorithms for some of these problems.

While work on finite-representations of counterexamples and witnesses in the qualitative setting is known [5], we are not aware of such work in the quantitative verification domain. This work can be interpreted as automata-theoretic arithmetic, which has been explored in regular real analysis [12].

2 Preliminaries

Definition 1 (Büchi automata [21]). A (finite-state) Büchi automaton is a tuple $\mathcal{A} = (S, \Sigma, \delta, \text{Init}, \mathcal{F})$, 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, $\text{Init} \subseteq S$ is the set of initial states, and $\mathcal{F} \subseteq S$ is the set of accepting states.

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 $|Init| = 1$. Otherwise, it is *nondeterministic*. For a word $w = w_0w_1 \dots \in \Sigma^\omega$, a *run* ρ of w is a sequence of states $s_0s_1 \dots$ s.t. $s_0 \in Init$, and $\tau_i = (s_i, w_i, s_{i+1}) \in \delta$ for all i . Let $inf(\rho)$ denote the set of states that occur infinitely often in run ρ . A run ρ is an *accepting run* if $inf(\rho) \cap \mathcal{F} \neq \emptyset$. A word w is an accepting word if it has an accepting run. Büchi automata are known to be closed under set-theoretic union, intersection, and complementation [21]. Languages accepted by these automata are called ω -regular languages.

Definition 2 (Weighted ω -automaton [10,20]). A weighted ω -automaton over infinite words is a tuple $\mathcal{A} = (\mathcal{M}, \gamma)$, where $\mathcal{M} = (S, \Sigma, \delta, Init, S)$ is a Büchi automaton, and $\gamma : \delta \rightarrow \mathbb{Q}$ is a weight function.

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 $\rho = s_0s_1 \dots$ of word $w = w_0w_1 \dots$ is given by $wt_\rho = n_0n_1n_2 \dots$ where $n_i = \gamma(s_i, w_i, s_{i+1})$ for all i . The *weight of a run* ρ is given by $f(wt_\rho)$, where $f : \mathbb{Q}^\omega \rightarrow \mathbb{R}$ is an *aggregate function*. We use $f(\rho)$ to denote $f(wt_\rho)$.

Here the *weight of a word* $w \in \Sigma^\omega$ in weighted ω -automata is defined as $wt_{\mathcal{A}}(w) = \sup\{f(\rho) | \rho \text{ is a run of } w \text{ in } \mathcal{A}\}$. It can also be defined as the infimum of the weight of all its runs. By convention, if word $w \notin \mathcal{A}$, $wt_{\mathcal{A}}(w) = 0$ [10].

Definition 3 (Quantitative inclusion). Given two weighted ω -automata P and Q with aggregate function f , the 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 is PSPACE-complete for limsup and liminf [10], and undecidable for limit-average [16]. For discounted-sum with integer discount-factor it is in EXPTIME [6,10], and decidability is unknown for rational discount-factors

Definition 4 (Incomplete-information quantitative games). 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, $\gamma : S \rightarrow \mathbb{N} \times \mathbb{N}$ is the weight function, and $f : \mathbb{N}^\omega \rightarrow \mathbb{R}$ 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 $s_0a_0s_1a_1 \dots$, 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 $o_0a_0o_1a_1 \dots$, 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 $\gamma(s_0)_i\gamma(s_1)_i \dots$, 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 $\alpha : O^* \rightarrow \Sigma$ since it only sees observations. Player P_0 follows strategy α if for all i , $a_i = \alpha(o_0 \dots o_i)$. 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 .

Definition 5 (Büchi pushdown automata [13]). A Büchi pushdown automaton (Büchi PDA) is a tuple $\mathcal{A} = (S, \Sigma, \Gamma, \delta, \text{Init}, Z_0, \mathcal{F})$, 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, $\text{Init} \subseteq S$ is a set of initial states, $Z_0 \in \Gamma$ is the start symbol.

A run ρ on a word $w = w_0 w_1 \dots \in \Sigma^\omega$ of a Büchi PDA $\mathcal{A}$ is a sequence of configurations $(s_0, \gamma_0), (s_1, \gamma_1) \dots$ satisfying (1) $s_0 \in \text{Init}$, $\gamma_0 = Z_0$, 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 $\text{inf}(\rho)$ be the set of states that occur infinitely often in state sequence $s_0 s_1 \dots$ of run ρ . A run ρ is an *accepting run* in Büchi PDA if $\text{inf}(\rho) \cap \mathcal{F} \neq \emptyset$. 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).

We introduce some notation. For an infinite sequence $A = (a_0, a_1, \dots)$, $A[i]$ denotes its i -th element. Abusing notation, we write $w \in \mathcal{A}$ and $\rho \in \mathcal{A}$ if w and ρ are an accepting word and an accepting run of $\mathcal{A}$ respectively.

For missing proofs and constructions, refer to the supplementary material.

3 Comparator Automata

Comparator automata (often abbreviated as *comparators*) are a class of automata that can read pairs of weight sequences synchronously and establish an equality or inequality relationship between these sequences. Formally, we define:

Definition 6 (Comparator automata). Let Σ be a finite set of rational numbers, and $f : \mathbb{Q}^\omega \rightarrow \mathbb{R}$ denote an aggregate function. A comparator automaton for aggregate function f is an automaton over the alphabet $\Sigma \times \Sigma$ that accepts a pair (A, B) of (infinite) weight sequences iff $f(A) R f(B)$, where R is an inequality or the equality relation.

From now on, unless mentioned otherwise, we assume that all weight sequences are bounded, natural number sequences. The boundedness assumption is justified since the set of weights forming the alphabet of a comparator is bounded. For all aggregate functions considered in this paper, the result of comparison of weight sequences is preserved by a uniform linear transformation that converts rational-valued weights into natural numbers; justifying the natural number assumption.

We explain comparators through an example. The *limit supremum* (limsup, in short) of a bounded, integer sequence A , denoted by $\text{LimSup}(A)$, is the largest integer that appears infinitely often in A . The *limsup comparator* is a Büchi automaton that accepts the pair (A, B) of sequences iff $\text{LimSup}(A) \geq \text{LimSup}(B)$.

The working of the limsup comparator is based on non-deterministically guessing the limsup of sequences A and B , and then verifying that $\text{LimSup}(A) \geq \text{LimSup}(B)$. Büchi automaton $\mathcal{A}_k$ (Fig. 1) illustrates the basic building block of the limsup comparator. Automaton $\mathcal{A}_k$ accepts pair (A, B) of number sequences iff $\text{LimSup}(A) = k$, and $\text{LimSup}(B) \leq k$, for integer k . To see why this is true, first note that all incoming edges 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 $\{0, 1, \dots, k\}$. 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 . Together these ensure that $\text{LimSup}(A) = k$, and $\text{LimSup}(B) \leq k$. The union of such automata $\mathcal{A}_k$ for $k \in \{0, 1, \dots, \mu\}$ for upper bound μ , results in the limsup comparator. The *limit infimum* (liminf, in short) of an integer sequence is the smallest integer that appears infinitely often in it; its comparator is similar.

When the comparator for an aggregate function is a Büchi automaton, we call it an ω -regular comparator. Likewise, when the comparator for an aggregate function is a Büchi pushdown automaton, we call it an ω -context-free comparator. As seen here, the limsup and liminf comparators are ω -regular. Later, we see that discounted-sum comparator and prefix-average comparator are ω -regular and ω -context-free respectively (Sects. 4 and 5). We call an aggregate function ω -regular when it has an ω -regular comparator for at least one inequality relation. Due to closure properties of Büchi automata, comparators for all inequality and equality relations of an ω -regular aggregate function are also ω -regular.

Fig. 1. State f_k is an accepting state. Automaton $\mathcal{A}_k$ accepts (A, B) iff $\text{LimSup}(A) = k$, $\text{LimSup}(B) \leq k$. $*$ denotes $\{0, 1, \dots, \mu\}$, $\leq m$ denotes $\{0, 1, \dots, m\}$

Fig. 2. Weighted automaton P

Fig. 3. Weighted automaton Q

Motivating Example. Let weighted ω -automata P and Q be as illustrated in Figs. 2 and 3. The word $w = a(ab)^\omega$ has two runs $\rho_1^P = q_1(q_2)^\omega$, $\rho_2^P = q_1(q_3)^\omega$

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

```

1: Input: Weighted automata  $P, Q$ , and  $\omega$ -regular comparator  $\mathcal{A}_f$  (Inequality  $\leq$ )
2: Output: True if  $P \subseteq_f Q$ , False otherwise
3:  $\hat{P} \leftarrow \text{AugmentWtAndLabel}(P)$ 
4:  $\hat{Q} \leftarrow \text{AugmentWtAndLabel}(Q)$ 
5:  $\hat{P} \times \hat{Q} \leftarrow \text{MakeProduct}(\hat{P}, \hat{Q})$ 
6:  $\text{DimProof} \leftarrow \text{Intersect}(\hat{P} \times \hat{Q}, \mathcal{A}_{\succeq})$ 
7:  $\text{Dim} \leftarrow \text{FirstProject}(\text{DimProof})$ 
8: return  $\hat{P} \equiv \text{Dim}$ 

```

in P , and four runs $\rho_1^Q = q_1(q_2)^\omega$, $\rho_2^Q = q_1(q_3)^\omega$, $\rho_3^Q = q_1q_1(q_2)^\omega$, $\rho_4^Q = q_1q_1(q_3)^\omega$ in Q . Their weight-sequences are $wt_1^P = 3, (0, 1)^\omega$, $wt_2^P = 2, (2, 0)^\omega$ in P , and $wt_1^Q = (2, 1)^\omega$, $wt_2^Q = (0, 2)^\omega$, $wt_3^Q = 1, 2, (2, 1)^\omega$, $wt_4^Q = 1, 0, (0, 2)^\omega$ in Q .

To determine if w has greater weight in P or in Q , compare aggregate value of weight-sequences of runs in P and Q . Take the comparator for aggregate function f that accepts a pair (A, B) of weight-sequence iff $f(A) \leq f(B)$. For $wt_P(w) \leq wt_Q(w)$, for every run ρ_i^P in P , there exists a run ρ_j^Q in Q s.t. (ρ_i^P, ρ_j^Q) is accepted by the comparator. This forms the basis for quantitative inclusion.

3.1 Quantitative Inclusion

InclusionReg (Algorithm 1) is an algorithm for quantitative inclusion for ω -regular aggregate functions. For weighted ω -automata P, Q , and ω -regular comparator $\mathcal{A}_f$, InclusionReg returns True iff $P \subseteq_f Q$. We assume $P \subseteq Q$ (qualitative inclusion) to avoid trivial corner cases.

Key Ideas. $P \subseteq_f Q$ holds if for every run ρ_P in P on word w , there exists a run ρ_Q in Q on the same word w such that $f(\rho_P) \leq f(\rho_Q)$. We refer to such runs of P by *diminished run*. Hence, $P \subseteq_f Q$ iff all runs of P are diminished.

InclusionReg constructs Büchi automaton Dim that consists of exactly the diminished runs of P . It returns True iff Dim contains all runs of P . To obtain Dim , it constructs Büchi automaton DimProof 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 $f(\rho_P) \leq f(\rho_Q)$. The ω -regular comparator for inequality $\leq$ for function f ensures $f(\rho_P) \leq f(\rho_Q)$. The projection of DimProof on runs of P results in Dim .

Algorithm Details. InclusionReg has three steps: (a). **Uniqueld** (Lines 3–4): Enables unique identification of runs in P and Q through *labels*. (b). **Compare** (Lines 5–7): Compares weight of runs in P with weight of runs in Q , and constructs Dim . (c). **DimEnsure** (Line 8): Ensures if all runs of P are diminished.

1. **Uniqueld:** 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 $\hat{\rho} = (a_0, n_0, l_0)(a_1, n_1, l_1) \cdots \in \hat{\mathcal{A}}$ iff run $\rho \in \mathcal{A}$ on word $a_0a_1 \dots$ with weight sequence $n_0n_1 \dots$. Labels ensure bijection between

runs in $\mathcal{A}$ and words in $\hat{\mathcal{A}}$. Words of $\hat{\mathcal{A}}$ have a single run in $\hat{\mathcal{A}}$.

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 3–4).

2. **Compare:** The output of this step is the Büchi automaton Dim , that contains the word $\hat{\rho} \in \hat{P}$ iff ρ is a diminished run in P (Lines 5–7).

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 $\tau_{\mathcal{A}} = (s_{\mathcal{A}}, (a, n_{\mathcal{A}}, l_{\mathcal{A}}), t_{\mathcal{A}})$ 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 $\hat{P} \times \hat{Q}$.

Intersect intersects the weight components of $\hat{P} \times \hat{Q}$ with comparator $\mathcal{A}_f$ (Line 6). The resulting automaton $DimProof$ 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 projection of $DimProof$ on the words of $\hat{P}$ returns Dim which contains the word $\hat{\rho}_P$ iff ρ_P is a diminished run in P (Line 7).

3. **DimEnsure:** $P \subseteq_f Q$ iff $\hat{P} \equiv Dim$ (qualitative equivalence) since $\hat{P}$ consists of all runs of P and Dim consists of all diminished runs of P (Line 8).

Lemma 1. *Given weighted ω -automata P and Q with an ω -regular aggregate function f . **InclusionReg**($P, Q, \mathcal{A}_f$) returns True iff $P \subseteq_f Q$.*

Further, **InclusionReg** is adapted for quantitative *strict-inclusion* $P \subset_f Q$ i.e. for all words w , $wt_P(w) < wt_Q(w)$ by taking the ω -regular comparator $\mathcal{A}_f$ that accepts (A, B) iff $f(A) < f(B)$. Similarly for quantitative equivalence $P \equiv_f Q$.

Complexity Analysis. All operations in **InclusionReg** until Line 7 are polytime operations in the size of weighted ω -automata P , Q and comparator $\mathcal{A}_f$. Hence, Dim is polynomial in size of P , Q and $\mathcal{A}_f$. Line 8 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 $\bar{P}$, $\bar{Q}$ by assigning a weight of 1 to each transition. Since all runs in $\bar{P}$, $\bar{Q}$ have the same weight sequence, weight of all words in $\bar{P}$ and $\bar{Q}$ is the same for any function f . It is easy to see $P \subseteq Q$ (qualitative inclusion) iff $\bar{P} \subseteq_f \bar{Q}$ (quantitative inclusion).

Theorem 1. *Let P and Q be weighted ω -automata and $\mathcal{A}_f$ be an ω -regular comparator. The complexity of the quantitative inclusion problem, quantitative strict-inclusion problem, and quantitative equivalence problem for ω -regular aggregate function f is PSPACE-complete.*

Theorem 1 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. Previously, finite-state representations of counterexamples have been useful in verification and synthesis in qualitative systems [5], and could be useful in quantitative settings as well. However, we are not aware of procedures for such representations in the quantitative settings. Here we show that a trivial extension of **InclusionReg** yields Büchi automata-representations for all counterexamples of the quantitative inclusion problem for ω -regular functions.

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

Theorem 2. *All counterexamples of the quantitative inclusion problem for an ω -regular aggregate function can be expressed by a Büchi automaton.*

3.2 Incomplete-Information Quantitative Games

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 $\alpha : O^* \rightarrow \Sigma$ 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 $A^* \rightarrow B$ 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 $s_{i_0}, \dots, s_{i_k}$ corresponds to the state s_{i_k} reached via $s_{\mathcal{I}}, s_{i_0}, \dots, s_{i_{k-1}}$. Consider now a node x corresponding to state s and labeled by an action σ . Then x has children $xs_1, \dots, xs_n$, 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 [19]. 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. *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 $|D_f| = |\mathcal{A}_f|^{O(|\mathcal{A}_f|)}$.*

Since, D_f is the deterministic parity automaton equivalent to $\mathcal{A}_f$, $|D_f| = |\mathcal{A}_f|^{O(|\mathcal{A}_f|)}$. 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.

4 Discounted-Sum Comparator

The discounted-sum of an infinite sequence A with discount-factor $d > 1$, denoted by $DS(A, d)$, is defined as $\sum_{i=0}^{\infty} A[i]/d^i$. The discounted-sum comparator (DS-comparator, in short) for discount-factor d , denoted by $\mathcal{A}_{\succ_{DS(d)}}$, accepts a pair (A, B) of weight sequences iff $DS(A, d) < DS(B, d)$. We investigate properties of the DS-comparator, and show that the DS-comparator is ω -regular for all integral discount-factors d , and cannot be ω -regular when $1 < d < 2$.

Theorem 4. *DS-comparator for rational discount-factor $1 < d < 2$ is not ω -regular.*

For discounted-sum automaton $\mathcal{A}$ with discount factor d , the *cut-point language* of $\mathcal{A}$ w.r.t. $r \in \mathbb{R}$ is defined as $L^{\geq r} = \{w \in L(\mathcal{A}) \mid DS(w, d) \geq r\}$. It is known that the cut-point language $L^{\geq 1}$ with discount-factor $1 < d < 2$ is not ω -regular [9]. One can show that if DS-comparator for discount-factor $1 < d < 2$ were ω -regular, then cut-point language $L^{\geq 1}$ is also ω -regular; thus proving Theorem 4.

We provide the construction of DS-comparator with integer discount-factor.

Key Ideas. The core intuition is that sequences bounded by μ can be converted to their value in base d via a finite-state transducer. Lexicographic comparison of the resulting sequences renders the desired result. Conversion of sequences to base d requires a certain amount of *book-keeping* by the transducer. Here we describe a direct method for book-keeping and lexicographic comparison.

For natural-number sequence A and integer discount-factor $d > 1$, $DS(A, d)$ can be interpreted as a value in base d i.e. $DS(A, d) = A[0] + \frac{A[1]}{d} + \frac{A[2]}{d^2} + \dots = (A[0].A[1]A[2]\dots)_d$ [12]. Unlike comparison of numbers in base d , the lexicographically larger sequence may not be larger in value. This occurs because (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 these challenges, 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 , we obtain a sequence C . Arithmetic in base d also results in sequence X of carry elements. Then, we see:

Lemma 2. *Let A, B, C, X be number 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 $DS(B, d) = DS(A, d) + DS(C, d)$.

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) C refers to a representation of value $DS(B, d) - DS(A, d)$ in base d , and X is the carry-sequence. Hence if A and B are bounded-integer sequences, not only are X and C bounded sequences, they can be constructed from a *fixed finite set of integers*:

Lemma 3. *Let $d > 1$ be an integer discount-factor. Let A and B be nonnegative integer sequences bounded by μ s.t. $DS(A, d) < DS(B, d)$. Let C and X be as constructed in Lemma 2. There exists at least one pair of integer-sequences C and X that satisfy the following two equations*

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

In Büchi automaton $\mathcal{A}_{\succ_{DS(d)}}$ (i) states are represented by (x, c) where x and c range over all possible elements of X and C , which are finite, (ii) a special start state s , (iii) transitions from the start state $(s, (a, b), (x, c))$ satisfy $a + c + x = b$ to replicate Eq. 1 (Lemma 2) at the 0-th index, (iv) 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 Eq. 2 (Lemma 2) at indexes $i > 0$, and (v) all (x, c) states are accepting. Lemma 2 ensures that $\mathcal{A}_{\succ_{DS(d)}}$ accepts (A, B) iff $DS(B, d) = DS(A, d) + DS(C, d)$.

However, $\mathcal{A}_{\succ_{DS(d)}}$ is yet to guarantee $DS(C, d) > 0$. For this, we include non-accepting states $(x, \perp)$, where x ranges over all possible (finite) elements of X . Transitions into and out of states $(x, \perp)$ satisfy Eqs. 1 or 2 (depending on whether transition is from start state s) where $\perp$ is treated as $c = 0$. Transition from $(x, \perp)$ -states to (x, c) -states occurs only if $c > 0$. Hence, any valid execution of (A, B) will be an accepting run only if the execution witnesses a non-zero value of c . Since C is a non-negative sequence, this ensures $DS(C, d) > 0$.

Construction. Let $\mu_C = \mu \cdot \frac{d}{d-1}$ and $\mu_X = 1 + \frac{\mu}{d-1}$. $\mathcal{A}_{\succ_{DS(d)}} = (S, \Sigma, \delta_d, Init, \mathcal{F})$

- $S = Init \cup \mathcal{F} \cup S_{\perp}$ where
 - $Init = \{s\}$, $\mathcal{F} = \{(x, c) \mid |x| \leq \mu_X, 0 \leq c \leq \mu_C\}$, and
 - $S_{\perp} = \{(x, \perp) \mid |x| \leq \mu_X\}$ where $\perp$ is a special character, and $c \in \mathbb{N}$, $x \in \mathbb{Z}$.
- $\Sigma = \{(a, b) : 0 \leq a, b \leq \mu\}$ where a and b are integers.
- $\delta_d \subset S \times \Sigma \times S$ 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 5. *The DS-comparator with maximum bound μ , is ω -regular for integer discount-factors $d > 1$. Size of the discounted-sum comparator is $\mathcal{O}(\frac{\mu^2}{d})$.*

DS-comparator with non-strict inequality $\leq$ and equality $=$ follow similarly. Consequently, properties of ω -regular comparators hold for DS-comparator with integer discount-factor. Specifically, DS-inclusion is PSPACE-complete in size of the input weighted automata and DS-comparator. Since, size of DS-comparator is polynomial w.r.t. to upper bound μ (in unary), DS-inclusion is PSPACE in size of input weighted automata and μ . Not only does this bound improve upon the previously known upper bound of EXPTIME but it also closes the gap between upper and lower bounds for DS-inclusion.

Corollary 1. *Given weighted automata P and Q , maximum weight on their transitions μ in unary form and integer discount-factor $d > 1$, the DS-inclusion, DS-strict-inclusion, and DS-equivalence problems are PSPACE-complete.*

As mentioned earlier, the known upper bound for discounted-sum inclusion with integer discount-factor is exponential [6, 10]. This bound is based on an exponential determinization construction (subset construction) combined with arithmetical reasoning. 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 automata.

5 Limit-Average Comparator

The limit-average of an infinite sequence M is the point of convergence of the average of prefixes of M . Let $\text{Sum}(M[0, n-1])$ denote the sum of the n -length prefix of sequence M . The *limit-average infimum*, denoted by $\text{LimInfAvg}(M)$, is defined as $\liminf_{n \rightarrow \infty} \frac{1}{n} \cdot \text{Sum}(M[0, n-1])$. Similarly, the *limit-average supremum*, denoted by $\text{LimSupAvg}(M)$, is defined as $\limsup_{n \rightarrow \infty} \frac{1}{n} \cdot \text{Sum}(M[0, n-1])$. The limit-average of sequence M , denoted by $\text{LimAvg}(M)$, is defined *only if* the limit-average infimum and limit-average supremum coincide, and then $\text{LimAvg}(M) = \text{LimInfAvg}(M) (= \text{LimSupAvg}(M))$. Note that while limit-average infimum and supremum exist for all bounded sequences, the limit-average may not.

In existing work, limit-average is defined as the limit-average infimum (or limit-average supremum) to ensure that limit-average exists for all sequences [7, 10, 11, 22]. While this definition is justified in context of the application, it may lead to a misleading comparison in some cases. For example, consider

sequence A s.t. $\text{LimSupAvg}(A) = 2$ and $\text{LimInfAvg}(A) = 0$, and sequence B s.t. $\text{LimAvg}(B) = 1$. Clearly, limit-average of A does not exist. Suppose, $\text{LimAvg}(A) = \text{LimInfAvg}(A) = 0$, then $\text{LimAvg}(A) < \text{LimAvg}(B)$, deluding that average of prefixes of A are always less than those of B in the limit. This is untrue since $\text{LimSupAvg}(A) = 2$.

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.

5.1 Limit-Average Language and Comparison

Let $\Sigma = \{0, 1, \dots, \mu\}$ 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 [13], a similar argument proves that $\mathcal{L}_{LA}$ is not ω -context-free.

Quantifiers $\exists^\infty i$ and $\exists^f i$ denote the existence of *infinitely* many and *only finitely* many indices i , respectively.

Theorem 6. $\mathcal{L}_{LA}$ is neither an ω -regular nor an ω -context-free language.

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:

Lemma 4. Let A and B be sequences s.t. their limit average exists. If $\exists^\infty i, \text{Sum}(A[0, i-1]) \geq \text{Sum}(B[0, i-1])$ then $\text{LimAvg}(A) \geq \text{LimAvg}(B)$.

Lemma 5. Let A, B be sequences s.t their limit-average exists. If $\text{LimAvg}(A) > \text{LimAvg}(B)$ then $\exists^f i, \text{Sum}(B[0, i-1]) \geq \text{Sum}(A[0, i-1])$ and $\exists^\infty i, \text{Sum}(A[0, i-1]) > \text{Sum}(B[0, i-1])$.

5.2 Prefix-Average Comparison and Comparator

The previous section relates limit-average comparison with the sums of equal length prefixes of the sequences (Lemmas 4 and 5). 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 5). Specifically, for sequences A and B with equal limit-average it is possible that $\exists^\infty i, \text{Sum}(A[0, n-1]) > \text{Sum}(B[0, n-1])$ and $\exists^\infty i, \text{Sum}(B[0, n-1]) > \text{Sum}(A[0, n-1])$. 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 7 (Prefix-average comparison). *Let A and B be number sequences. We say $\text{PrefixAvg}(A) \geq \text{PrefixAvg}(B)$ if $\exists^f i, \text{Sum}(B[0, i-1]) \geq \text{Sum}(A[0, i-1])$ and $\exists^\infty i, \text{Sum}(A[0, i-1]) > \text{Sum}(B[0, i-1])$.*

Intuitively, prefix-average comparison states that $\text{PrefixAvg}(A) \geq \text{PrefixAvg}(B)$ 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. Definition 7 and Lemmas 4, 5 relate limit-average comparison and prefix-average comparison:

Corollary 2. *When limit-average of A and B exists, then*

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

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 $\text{LimInfAvg}(A) > \text{LimSupAvg}(B)$, then $\text{PrefixAvg}(A) \geq \text{PrefixAvg}(B)$. Therefore, prefix-average comparison relaxes limit-average comparison.

The rest of this section describes *prefix-average comparator* $\mathcal{A}_{\succeq_{PA}(\cdot)}$, an automaton that accepts the pair (A, B) of sequences iff $\text{PrefixAvg}(A) \geq \text{PrefixAvg}(B)$.

Lemma 6 (Pumping Lemma for ω -regular language [2]). *Let L be an ω -regular language. There exists $p \in \mathbb{N}$ such that, for each $w = u_1 w_1 u_2 w_2 \dots \in L$ 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 $u_1 x_1 y_1^{j_1} z_1 u_2 x_2 y_2^{j_2} z_2 \dots \in L$.*

Theorem 7. *The prefix-average comparator is not ω -regular.*

Proof (Proof Sketch). We use Lemma 6 to prove that $\mathcal{A}_{\succeq_{PA}(\cdot)}$ is not ω -regular. Suppose $\mathcal{A}_{\succeq_{PA}(\cdot)}$ 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 $\mathcal{L}_{\succeq_{PA}(\cdot)}$.

Concretely, $A = (0^p 1^{2p})^\omega$, $B = (1^p 0^{2p})^\omega$, $\text{LimAvg}(A) = \frac{2}{3}$, $\text{LimAvg}(B) = \frac{1}{3}$. So, $w = (A, B) \in \mathcal{A}_{\succeq_{PA}(\cdot)}$. Select as factor w_i (from Lemma 6) 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 $\hat{w} = (\hat{A}, \hat{B}) \notin \mathcal{L}_{\succeq_{PA}(\cdot)}$.

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

Key Ideas. For sequences A and B to satisfy $\text{PrefixAvg}(A) \geq \text{PrefixAvg}(B)$, $\exists^f i, \text{Sum}(B[0, i-1]) \geq \text{Sum}(A[0, i-1])$ and $\exists^\infty i, \text{Sum}(A[0, i-1]) > \text{Sum}(B[0, i-1])$. This occurs iff there exists an index N s.t. for all indices $i > N$, $\text{Sum}(A[0, i-1]) - \text{Sum}(B[0, i-1]) > 0$. 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 $\mathcal{A}_{\succeq_{PA}(\cdot)}$ 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. $\mathcal{A}_{\succeq_{PA}(\cdot)}$ 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 $|\text{Sum}(A[0, i-1]) - \text{Sum}(B[0, i-1])|$. Furthermore, in state s_N , $\text{Sum}(A[0, i-1]) - \text{Sum}(B[0, i-1]) \leq 0$, and in state s_P and s_F , $\text{Sum}(A[0, i-1]) - \text{Sum}(B[0, i-1]) > 0$. 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 $\text{Sum}(A[0, i-1]) - \text{Sum}(B[0, i-1]) > 0$.

Theorem 8. *The prefix-average comparator 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.

6 Conclusion

In this paper, we 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 show that the discounted-sum inclusion problem is PSAPCE-complete for integer discount-factor, hence closing a complexity gap. We also studied the discounted-sum and prefix-average comparator, which are ω -regular and ω -context-free, respectively.

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 develop a clearer understanding of when aggregate functions are ω -regular.

Acknowledgements. We thank the anonymous reviewers for their comments. We thank K. Chatterjee, L. Doyen, G. A. Perez and J. F. Raskin for corrections to earlier drafts, and their contributions to this paper. We thank P. Ganty and R. Majumdar for preliminary discussions on the limit-average comparator. This work was partially supported by NSF Grant No. 1704883, “Formal Analysis and Synthesis of Multiagent Systems with Incentives”.

References

1. Almagor, S., Boker, U., Kupferman, O.: What’s decidable about weighted automata? In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 482–491. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24372-1_37
2. Alur, R., Degorre, A., Maler, O., Weiss, G.: On omega-languages defined by mean-payoff conditions. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 333–347. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00596-1_24
3. Andersen, G., Conitzer, V.: Fast equilibrium computation for infinitely repeated games. In: Proceedings of AAAI, pp. 53–59 (2013)
4. Andersson, D.: An improved algorithm for discounted payoff games. In: ESSLLI Student Session, pp. 91–98 (2006)
5. Baier, C., Katoen, J.-P., et al.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Boker, U., Henzinger, T.A.: Exact and approximate determinization of discounted-sum automata. LMCS **10**(1) (2014)
7. Brim, L., Chaloupka, J., Doyen, L., Gentilini, R., Raskin, J.-F.: Faster algorithms for mean-payoff games. Formal Methods Syst. Des. **38**(2), 97–118 (2011)

8. Chatterjee, K., Doyen, L.: Energy parity games. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part II. LNCS, vol. 6199, pp. 599–610. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14162-1_50
9. Chatterjee, K., Doyen, L., Henzinger, T.A.: Expressiveness and closure properties for quantitative languages. In: Proceedings of LICS, pp. 199–208. IEEE (2009)
10. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. Trans. Comput. Log. **11**(4), 23 (2010)
11. Chatterjee, K., Henzinger, T.A., Jurdzinski, M.: Mean-payoff parity games. In: Proceedings of LICS, pp. 178–187. IEEE (2005)
12. Chaudhuri, S., Sankaranarayanan, S., Vardi, M.Y.: Regular real analysis. In: Proceedings of LICS, pp. 509–518 (2013)
13. Cohen, R.S., Gold, A.Y.: Theory of ω -languages: characterizations of ω -context-free languages. J. Comput. Syst. Sci. **15**(2), 169–184 (1977)
14. de Alfaro, L., Faella, M., Henzinger, T.A., Majumdar, R., Stoelinga, M.: Model checking discounted temporal properties. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 77–92. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_6
15. de Alfaro, L., Faella, M., Stoelinga, M.: Linear and branching metrics for quantitative transition systems. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 97–109. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27836-8_11
16. Degorre, A., Doyen, L., Gentilini, R., Raskin, J.-F., Toruńczyk, S.: Energy and mean-payoff games with imperfect information. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 260–274. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15205-4_22
17. Droste, M., Kuich, W., Vogler, H.: Handbook of Weighted Automata. Springer, Heidelberg (2009). <https://doi.org/10.1007/978-3-642-01492-5>
18. Karp, R.M.: A characterization of the minimum cycle mean in a digraph. Discret. Math. **23**(3), 309–311 (1978)
19. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete information. In: Barringer, H., Fisher, M., Gabbay, D., Gough, G. (eds.) Advances in Temporal Logic, pp. 109–127. Springer, Dordrecht (2000). https://doi.org/10.1007/978-94-015-9586-5_6
20. Mohri, M.: Weighted automata algorithms. In: Mohri, M. (ed.) Handbook of Weighted Automata, pp. 213–254. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01492-5_6
21. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata Logics, and Infinite Games: A Guide to Current Research. LNCS, vol. 2500. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-36387-4>
22. Zwick, U., Paterson, M.: The complexity of mean payoff games on graphs. Theor. Comput. Sci. **158**(1), 343–359 (1996)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.