Comparator Automata in Quantitative Verification

Suguman Bansal, Swarat Chaudhuri, and Moshe Y. Vardi

Rice University, Houston TX, TX 77005, USA

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 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* $f : \mathbb{Q}^\omega \to \mathbb{R}$, 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 [14,15,16]. 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 § 2. Comparator automata is formally defined in § 3. Generic algorithms for ω-regular comparators are discussed in § 3.1-3.2. The construction and properties of discounted-sum comparator, and limit-average and prefix-average comparator are given in § 4, and § 5, respectively. We conclude with future directions in § 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* $\mathcal{A} = (S, \Sigma, \delta, \mathit{Init}, \mathcal{F})$*, 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 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*. 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. 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* $\gamma : \delta \to \mathbb{Q}$ *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 $\rho = s_0 s_1 \dots$ 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 $f(wt_\rho)$, where $f : \mathbb{Q}^\omega \to \mathbb{R}$ 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 $wt_\mathcal{A}(w) = sup\{f(\rho) | \rho$ is a run of w 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*, $\gamma : S \to \mathbb{N} \times \mathbb{N}$ is the* weight function*, and $f : \mathbb{N}^\omega \to \mathbb{R}$ 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 $s_0 a_0 s_1 a_1 \dots$, 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 $o_0 a_0 o_1 a a_1 \dots$, 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 $\alpha : O^* \to \Sigma$ 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* $\mathcal{A} = (S, \Sigma, \Gamma, \delta, Init, Z_0, \mathcal{F})$*, 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*, $Init \subseteq S$ 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).

We introduce some notation. For an infinite sequence $A = (a_0, a_1, \ldots)$, $A[i]$ 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 $f : \mathbb{Q}^\omega \to \mathbb{R}$ 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 $\mathsf{LimSup}(\mathsf{A})$, 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 $\mathsf{LimSup}(\mathsf{A}) \geq \mathsf{LimSup}(\mathsf{B})$.

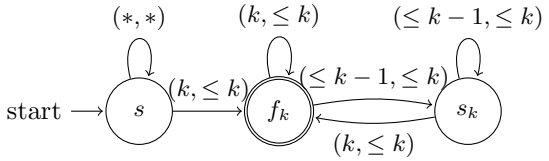

Fig. 1. State f_k is an accepting state. Automaton $\mathcal{A}_k$ accepts (A, B) iff $\mathsf{LimSup}(\mathsf{A}) = k$, $\mathsf{LimSup}(\mathsf{B}) \leq k$. $*$ denotes $\{0, 1 \ldots \mu\}$, $\leq m$ denotes $\{0, 1 \ldots, m\}$

The working of the limsup comparator is based on non-deterministically guessing the limsup of sequences A and B, and then verifying that $\mathsf{LimSup}(\mathsf{A}) \geq \mathsf{LimSup}(\mathsf{B})$. 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 $\mathsf{LimSup}(\mathsf{A}) = k$, and $\mathsf{LimSup}(\mathsf{B}) \leq k$, for integer k. To see why this is true, first note that all incoming edges to accepting state f_k occur on alphabet $(k, \leq k)$

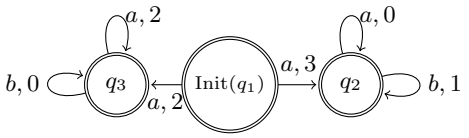

Fig. 2. Weighted automaton P

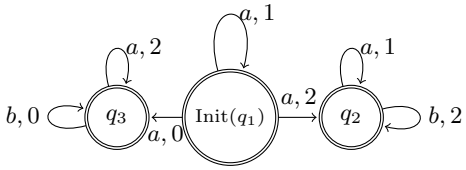

Fig. 3. Weighted automaton Q

while all transitions between states f_k and s_k occur on alphabet $(\leq k, \leq k)$, where $\leq k$ denotes the set $\{0, 1, \ldots k\}$. 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. Together these ensure that $\mathsf{LimSup}(\mathsf{A}) = k$, and $\mathsf{LimSup}(\mathsf{B}) \leq k$. The union of such automata $\mathcal{A}_k$ for $k \in \{0, 1, \ldots \mu\}$ 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 (§ 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.

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

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

1: **Input:** Weighted automata P, Q, and ω-regular comparator $\mathcal{A}_f$ (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: *DimProof* $\leftarrow$ Intersect$(\hat{P} \times \hat{Q}, \mathcal{A}_{\geq})$
7: *Dim* $\leftarrow$ FirstProject(*DimProof*)
8: **return** $\hat{P} \equiv$ *Dim*

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). UniqueId (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. 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 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 $\hat{\tau_{\mathcal{A}}} = (s_{\mathcal{A}}, (a, n_{\mathcal{A}}, l_{\mathcal{A}}), t_{\mathcal{A}})$ 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 $\hat{P} \times \hat{Q}$.
   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 $\overline{P}$, $\overline{Q}$ 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 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 $\hat{P} \setminus \mathit{Dim}$. The counterexamples words can be obtained from $\hat{P} \setminus \mathit{Dim}$ 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 $\alpha : O^* \to \Sigma$ 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 [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 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 $\Sigma_{i=0}^{\infty} A[i]/d^i$. 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 $L^{\geq r} = \{w \in L(\mathcal{A}) | DS(w, d) \geq r\}$. 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. $DS(A, d) = A[0] + \frac{A[1]}{d} + \frac{A[2]}{d^2} + \cdots = (A[0].A[1]A[2]\dots)_d$ [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 Equation 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 Equation 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 Equation 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
  $Init=\{s\}$, $\mathcal{F}=\{(x,c)||x|\leq\mu_X,0\leq c\leq\mu_C\}$, and
  $S_\perp=\{(x,\perp)||x|\leq\mu_X\}$ where $\perp$ is a special character, and $c\in\mathbb{N}$, $x\in\mathbb{Z}$.
- $\Sigma=\{(a,b):0\leq a,b\leq\mu\}$ 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 $\mathsf{Sum}(M[0, n-1])$ denote the sum of the n-length prefix of sequence M. The *limit-average infimum*, denoted by $\mathsf{LimInfAvg}(M)$, is defined as $\liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(M[0, n-1])$. Similarly, the *limit-average supremum*, denoted by $\mathsf{LimSupAvg}(M)$, is defined as $\limsup_{n\to\infty} \frac{1}{n} \cdot \mathsf{Sum}(M[0, n-1])$. The limit-average of sequence M, denoted by $\mathsf{LimAvg}(M)$, is defined *only if* the limit-average infimum and limit-average supremum coincide, and then $\mathsf{LimAvg}(M) = \mathsf{LimInfAvg}(M)$ $(= \mathsf{LimSupAvg}(M))$. 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. $\mathsf{LimSupAvg}(A) = 2$ and $\mathsf{LimInfAvg}(A) = 0$, and sequence B s.t. $\mathsf{LimAvg}(B) = 1$. Clearly, limit-average of A does not exist. Suppose, $\mathsf{LimAvg}(A) = \mathsf{LimInfAvg}(A) = 0$, then $\mathsf{LimAvg}(A) < \mathsf{LimAvg}(B)$, deluding that average of prefixes of A are always less than those of B in the limit. This is untrue since $\mathsf{LimSupAvg}(A) = 2$.

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 $\Sigma = \{0, 1, \ldots, \mu\}$ 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 [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 $\exists^\infty i, \mathsf{Sum}(A[0, i-1]) \geq \mathsf{Sum}(B[0, i-1])$ then $\mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$.*

Lemma 5. *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 $\exists^\infty i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$.*

5.2 Prefix-average comparison and comparator

The previous section relates limit-average comparison with the sums of equal length prefixes of the sequences (Lemma 4-5). 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 $\exists^\infty i, \mathsf{Sum}(A[0, n-1]) > \mathsf{Sum}(B[0, n-1])$ 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 7 (Prefix-average comparison). *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 $\exists^\infty i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$.*

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. Definition7 and Lemma 4-5 relate limit-average comparison and prefix-average comparison:

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

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

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 $\mathsf{LimInfAvg}(A) > \mathsf{LimSupAvg}(B)$, 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* $\mathcal{A}_{\succeq_{PA(\cdot)}}$, an automaton that accepts the pair (A, B) of sequences iff $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.

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 $w = u_1 w_1 u_2 w_2 \cdots \in L$ 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 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, $A = (0^p 1^{2p})^\omega$, $B = (1^p 0^{2p})^\omega$, $\mathsf{LimAvg}(A) = \frac{2}{3}$, $\mathsf{LimAvg}(B) = \frac{1}{3}$. 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 $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1])$, 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 $\mathcal{A}_{\succeq_{PA(\cdot)}}$ 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 $|\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1])|$. 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 $\mathsf{Sum}(A[0, i-1]) - \mathsf{Sum}(B[0, i-1]) > 0$.

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. S. Almagor, U. Boker, and O. Kupferman. Whatś decidable about weighted automata? In *Proc. of ATVA*, pages 482–491. Springer, 2011.
2. R. Alur, A. Degorre, O. Maler, and G. Weiss. On omega-languages defined by mean-payoff conditions. In *Proc. of FOSSACS*, pages 333–347. Springer, 2009.
3. G. Andersen and V. Conitzer. Fast equilibrium computation for infinitely repeated games. In *Proc. of AAAI*, pages 53–59, 2013.
4. D. Andersson. An improved algorithm for discounted payoff games. In *ESSLLI Student Session*, pages 91–98, 2006.
5. C. Baier, J.-P. Katoen, et al. *Principles of model checking*. MIT press Cambridge, 2008.
6. U. Boker and T. A. Henzinger. Exact and approximate determinization of discounted-sum automata. *LMCS*, 10(1), 2014.
7. L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.
8. K. Chatterjee and L. Doyen. Energy parity games. In *In Proc. of ICALP*, pages 599–610. Springer, 2010.
9. K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. In *Proc. of LICS*, pages 199–208. IEEE, 2009.
10. K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *Transactions on Computational Logic*, 11(4):23, 2010.
11. K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-payoff parity games. In *Proc. of LICS*, pages 178–187. IEEE, 2005.
12. S. Chaudhuri, S. Sankaranarayanan, and M. Y. Vardi. Regular real analysis. In *Proc. of LICS*, pages 509–518, 2013.
13. R. S. Cohen and A. Y. Gold. Theory of ω-languages: Characterizations of ω-context-free languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977.
14. 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.
15. 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.
16. A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Toruńczyk. Energy and mean-payoff games with imperfect information. In *International Workshop on Computer Science Logic*, pages 260–274. Springer, 2010.
17. M. Droste, W. Kuich, and H. Vogler. *Handbook of weighted automata*. Springer, 2009.
18. R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.
19. O. Kupferman and M. Y. Vardi. Synthesis with incomplete informatio. In *Advances in Temporal Logic*, pages 109–127. Springer, 2000.
20. M. Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.
21. W. Thomas, T. Wilke, et al. *Automata, logics, and infinite games: A guide to current research*, volume 2500. Springer Science & Business Media, 2002.
22. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343–359, 1996.

A Limit supremum comparator

Lemma 1. *Let A and B be non-negative integer sequences bounded by μ.*

Büchi automaton $\mathcal{A}_k$ (Fig. 1) accepts (A, B) iff $\mathsf{LimSup(A)} = k$, *and* $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$.

Proof. For any integer sequence A, the limsup refers to the maximum integer that occurs infinitely often in A. Hence if $\mathsf{LimSup(A)} = k$, then 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 any integer greater than k in A.

Following the above argument, for integer sequences A and B, when $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$, beyond index $l = max(l_A, l_B)$, integers greater than k will not occur. Büchi automaton $\mathcal{A}_k$ (Fig. 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).

On the other hand, if a run of (A, B) is accepted by the automaton, then the run must have visited state f_k infinitely often. Since all incoming transitions to state f_k occur on the alphabet (k, b) for some integer b, k must occur infinitely often in A. Furthermore, if a run visits state f_k once, it only visits state f_k and s_k thereon. All transitions from these states occur on alphabet (a, b) where $a, b \leq k$. Therefore, all infinitely occurring digits in A and B must be less than or equal to k. With these two observations, we infer that if (A, B) has an accepting run, then $\mathsf{LimSup(A)} = k$, and $\mathsf{LimSup(A)} \geq \mathsf{LimSup(B)}$.

A.1 Construction of limsup comparator

Suppose all sequences are natural number sequences, bounded by μ. The limsup comparator is the Büchi automaton $\mathcal{A}_{\succ LS} = (S, \Sigma, \delta, Init, \mathcal{F})$ where

- $S = \{s\} \cup \{s_0, s_1 \ldots, s_\mu\} \cup \{f_0, f_1 \ldots, f_\mu\}$
- $\Sigma = \{(a, b) : 0 \leq a, b \leq \mu\}$ where a and b are integers.
- $\delta \subseteq S \times \Sigma \times S$ is defined as follows:
  1. Transitions from start state s: $(s, (a, b), p)$ for all $(a, b) \in \Sigma$, and for all $p \in \{s\} \cup \{f_0, f_1, \ldots, f_\mu\}$.
  2. Transitions between f_k and s_k for each k:
     - i (f_k, α, f_k) for $\alpha \in \{k\} \times \{0, 1, \ldots k\}$.
     - ii (f_k, α, s_k) for $\alpha \in \{0, 1, \ldots k-1\} \times \{0, 1, \ldots k\}$.
     - iii (s_k, α, s_k) for $\alpha \in \{0, 1, \ldots k-1\} \times \{0, 1, \ldots k\}$.
     - iv (s_k, α, f_k) for $\alpha \in \{k\} \times \{0, 1, \ldots k\}$.
- $Init = \{s\}$
- $\mathcal{F} = \{f_0, f_1 \ldots, f_\mu\}$

B Quantitative Inclusion for ω-regular aggregate functions

In this section, we prove that $\mathsf{InclusionReg}(P, Q, \mathcal{A}_f)$ (Algorithm 1) returns True iff $P \subseteq_f Q$.

Lemma 2. *Büchi automaton Dim consists of all diminished runs of weighted automata P.*

In other words, if ρ is a diminished run in P on word $w_0w_1\ldots$ with weight sequence $n_0n_1\ldots$, then $\hat{\rho} = (w_0, n_0, l_0)(w_1, n_1, l_1)\ldots$ is a word in Dim where $l_0l_1\ldots$ is a unique identifier for the run in P.

Proof. Let $\mathcal{A}_f$ be the comparator for ω-regular aggregate function f 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 $(\hat{\rho}, \hat{\sigma}) \in \hat{P} \times \hat{Q}$, then $\rho \in P$, $\sigma \in Q$ and word on ρ and σ in P and Q respectively is the same. Next, $\hat{P} \times \hat{Q}$ is intersected over the weight sequences with $\mathcal{A}_f$. Since $\mathcal{A}_f$ accepts (A, B) iff $f(A) \leq f(B)$, $(\hat{\rho}, \hat{\sigma}) \in \hat{P} \times \hat{Q} \cap^* \mathcal{A}_f$ iff weight sequence of ρ and σ, wt_ρ and wt_σ respectively, are related as $f(wt_\rho) \leq f(wt_\sigma)$. 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 *Dim* constitutes of these $\hat{\rho}$.

Therefore *Dim* consists of $\hat{\rho}$ only if ρ is a diminished run in P.

Every step of the algorithm has a two-way implication, hence t is also true that every diminished run of P is present in *Dim*.

Lemma 3. *Given weighted automata P and Q with an ω-regular aggregate function.*

$\mathsf{InclusionReg}(P, Q, \mathcal{A}_f)$ *(Algorithm 1) returns* True *iff $P \subseteq_f Q$.*

Proof. $\hat{P}$ consists of all runs of P. *Dim* consists of all diminished runs of P. $P \subseteq_f Q$ iff every run of P is a diminished run. Therefore the solution to the quantitative inclusion problem is given by whether $\hat{P} \equiv Dim$, where $\equiv$ denotes qualitative equivalence.

B.1 Finite-state representation for Counterexample automata

Counterexample automata We refer to the counterexample automata by *NotDim* here. The key idea is to construct a Büchi automaton for (i) all runs of weighted ω-automata P that are not diminished runs. (ii) words (qualitatively) contained in P that are not (qualitatively) contained in Q. These ideas are expanded below:

1. Recall from Algorithm 1, word $\hat{\rho}$ is contained in *Dim* iff ρ is a diminished run in weighted automaton P. From Büchi automaton $\hat{P}$ and *Dim* we construct

the immediate automaton $\mathcal{I} = \hat{P} \setminus Dim$ (qualitative setminus). Automaton $\mathcal{I}$ contains $\hat{\rho}$ iff ρ is not a diminished run in P.
Counterexamples are extracted from $\mathcal{I}$ by simply converting every transition $\hat{\tau} = (s, (a, wt, l), t)$ in $\mathcal{I}$ to transition $\tau : (s, (a, wt), t)$ in *NotDim*.
It is easy to prove that Büchi automaton *NotDim* accepts (w, A) if it is a counterexample of the quantitative inclusion problem $P \subseteq_f Q$ where f is an ω-regular aggregate function.

2. The other counterexamples are computed as $P \setminus Q$ (qualitative setminus).

Büchi automaton $CounterExample = NotDim \cup (P \setminus Q)$ contains all counterexamples of quantitative inclusion for ω-regular aggregate function f.

C Discounted Sum Comparator

Theorem 1. *There does not exist an ω-regular discounted-sum comparator for non-integer discount-factor $1 < d < 2$.*

Proof. For discounted-sum automaton $\mathcal{A}$ with discount factor d, the cut-point language of $\mathcal{A}$ with w.r.t. $r \in \mathbb{R}$ is defined as $L^{\geq r} = \{w \in L(\mathcal{A}) | DS(w, d) \geq r\}$.

It has been shown that for all deterministic discounted-sum automata with discount factor $1 < d < 2$, the cut-point language $L^{\geq 1}$ cannot be ω-regular [9]. .

If suppose there exists an ω-regular comparator $\mathcal{A}_d$ for rational discount factor $1 < d < 2$, then the cut-point language $L^{\geq 1}$ of deterministic disocunted-sum automata A can be constructed by taking product of $\mathcal{A} \times (1, 0, 0 \dots)$, and then intersection with $\mathcal{A}_d$. Since all actions are closed under ω-regular operations, $L^{\geq 1}$ can be represented with a Büchi automata.

This contradicts the result about ω-regularity of cut-point languages [9]. .

Theorem 2. *Let $d > 1$ be a positive integer discount-factor, and $\mu \in \mathbb{Z}^+$. There exists a Büchi automaton $\mathcal{A}_{\succ_{DS(d)}}$ such that $(A, B) \in \mathcal{A}_{\succ_{DS(d)}}$ if and only if $DS(A, d) > DS(B, d)$ for all non-negative integer sequences A and B bounded by μ.*

We provide a construction for $\mathcal{A}_{\succ_{DS(d)}}$. We show that (A, B) is accepted by $\mathcal{A}_{\succ_{DS(d)}}$ if and only if $DS(A, d) > DS(B, d)$.

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

- $S = \{s\} \cup \mathcal{F} \cup S_0$ where $\mathcal{F} = \{(x, c) || x| \leq \mu_X, 0 \leq c \leq \mu_C\}$, and $S_0 = \{(x, \perp) || x| \leq \mu_X\}$ where x and c integers.
- $\Sigma = \{(a, b) : 0 \leq a, b \leq \mu\}$ where a and b are integers.
- δ_d is defined as follows:
  1. Transitions from start state s
     (a) $s \xrightarrow{(a,b)} x$ for all $(x, \perp) \in S_0$ s.t. $b + x = a$
     (b) $s \xrightarrow{(a,b)} (x, c)$ for all $(x, c) \in \mathcal{F}$ s.t. $b + x + c = a$ and $c \neq 0$.
  2. Transitions within S_0: $(x, \perp) \xrightarrow{(a,b)} (x', \perp)$ for all $(x', \perp) \in S_0$, if $b + x' = a + d \cdot x$

3. Transitions within $\mathcal{F}$: $(x,c) \xrightarrow{(a,b)} (x',c')$ for all $(x',c') \in \mathcal{F}$ where $c' < d$, if $b + x' + c' = a + d \cdot x$
4. Transition between S_0 and $\mathcal{F}$: $(x,\perp) \xrightarrow{(a,b)} (x',c')$ for all $(x',c') \in \mathcal{F}$ where $0 < c' < d$, if $b + x' + c' = a + d \cdot x$

– $Init = \{s\}$
– $\mathcal{F} = \mathcal{F}$.

C.1 (A,B) has accepting run $\implies DS(A,d) > DS(B,d)$

Lemma 4. *Let A, B, C, X be the number sequences, $d => 1$ be a positive integer such that following invariant holds true:*

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

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

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

Lemma 5. *For every infinite run of (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$, we can construct sequence C, s.t sequences $DS(A,d) = DS(B,d) + DS(C,d)$.*

Proof. Define functions $XVal$ and $CVal : S \mapsto \mathbb{Z}$ where S is the set of all states in $\mathcal{A}_{\succ_{DS(d)}}$ as follows:

$$CVal(state) = \begin{cases} 0 & \text{if } state = s \in Init \\ c & \text{if } state = (x,c) \in \mathcal{F} \\ 0 & \text{if } state = (x,\perp) \in S_0 \end{cases}$$

$$XVal(state) = \begin{cases} 0 & \text{if } state = s \in Init \\ x & \text{if } state = (x,c) \in \mathcal{F} \\ x & \text{if } state = (x,\perp) \in S_0 \end{cases}$$

Consider an infinite run of word (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$. Let $\{state_i\}_i$ be the sequence of states visited by this run. Construct sequences C and X s.t. $C[i] = CVal(state_{i+1})$ and $X[i] = XVal(state_{i+1})$.

When $i = 0$, then either $X[0]$ and $C[0]$ derive their values from either $(x,c) \in \mathcal{F}$, or from $(x,\perp) \in S_0$. We consider the cases separately below:

1. From $(x,c) \in \mathcal{F}$: In this case, the 0-th transition is $s \xrightarrow{(a,b)} (x,c)$. From construction, we know that $b+x+c = a$. Since, $A[0] = a$, $B[0] = b$, $C[0] = c$, and $X[0] = x$, we have $B[0] + X[0] + C[0] = A[0]$.
2. From $(x,\perp) \in S_0$: In this case, the 0-th transition is $s \xrightarrow{(a,b)} (x,\perp)$. From construction, we know that $b + x = a$. Since, $A[0] = a$, $B[0] = b$, $C[0] = 0$, and $X[0] = x$, we have $B[0] + X[0] + C[0] = A[0]$.

Therefore, in either case, the first condition on Lemma 4 is satisfied.

For $i \geq 1$, transitions are either within S_0, or within $\mathcal{F}$, or from S_0 to $\mathcal{F}$. We will consider each of these cases separately:

1. Within S_0: Transitions are of the form $(x, \perp) \xrightarrow{(a,b)} (x', \perp)$ where $b + x' = a + d \cdot x$. Suppose this is the i-th transition in a run (for $i \geq 1$). In this case, $A[i] = a$, $B[i] = b$, $C[i] = CVal(x') = 0$, $X[i] = XVal(x') = x'$, and $X[i-1] = XVal(x) = x$. So, $A[i] + d \cdot X[i-1] = B[i] + X[i] + C[i]$.
2. Within $\mathcal{F}$: Transitions are of the form of $(x, c) \xrightarrow{(a,b)} (x', c')$. Suppose this is the ith transition in a run (for $i \geq 1$). From construction we know that $b + x' + c' = a + d \cdot x$. Here $A[i] = a$, $B[i] = b$, $C[i] = c$, $X[i] = x$, and $X[i-1] = x'$. Therefore, $A[i] + d \cdot X[i-1] = B[i] + X[i] + C[i]$.
3. From S_0 to $\mathcal{F}$: Transitions are of the form $(x, \perp) \xrightarrow{(a,b)} (x', c')$ s.t. $b + x' + c' = a + d \cdot x$. Suppose this is the i-th transition, then $A[i] = a$, $B[i] = b$, $X[i-1] = x$, $X[i] = x'$, $C[i] = c'$. Therefore, $A[i] + d \cdot X[i-1] = B[i] + X[i] + C[i]$.

In each of these cases, when $i \geq 1$, the second condition in Lemma 4 is satisfied.

Together we see that, for any infinite run of (A, B), we can construct sequences C and X, s.t. using Lemma 4 we can prove that $DS(A, d) = DS(B, d) + DS(C, d)$.

Lemma 6. *For every accepting run of (A, B) in $\mathcal{A}_{\succ_{DS(d)}}$, $DS(C, d) > 0$.*

Proof. For a run to be accepting in $\mathcal{A}_{\succ_{DS(d)}}$, it must visit at least one state in $\mathcal{F}$ infinitely often. Note that once a run visits a state in $\mathcal{F}$, it only visits states in $\mathcal{F}$ since there are no transitions out of $\mathcal{F}$. Since there are finitely many $\mathcal{F}$, the sufficient condition for a state to be accepting is that a run must enter $\mathcal{F}$. There are two ways of entering $\mathcal{F}$, either via $s \xrightarrow{(a,b)} (x, c)$ or via $(x, \perp) \xrightarrow{(a,b)} (x', c')$. Suppose this is the i-th transition in the accepting run, then $C[i] \neq 0$. Also, we know that for all j, $C[j] \geq 0$. Hence, $DS(C, d) \geq C[i] > 0$.

Corollary 1. *For all accepting runs of (A, B) in $\mathcal{A}_{\succ_{DS(d)}}$, $DS(A, d) > DS(B, d)$.*

Proof. From Lemma 5, and Lemma 6, it is clear that for all accepting runs of (A, B) in $\mathcal{A}_{\succ_{DS(d)}}$, $DS(A, d) > DS(B, d)$.

C.2 $DS(A, d) > DS(B, d) \implies (A, B)$ has an accepting run

We want to show that for non-negative bounded sequences A, B s.t. $DS(A, d) \geq DS(B, d)$, there exists an infinite run of (A, B) in $\mathcal{A}_{\succ_{DS(d)}}$. For the rest of this section, we will let A and B be bounded by μ. Let $DS^-(A, B, d, i) = \Sigma_{j=0}^{i}(A[j] - B[j]) \cdot \frac{1}{d^j}$. Also, we let $DS^-(A, B, d, \cdot) = \Sigma_{j=0}^{\infty}(A[j] - B[j]) \cdot \frac{1}{d^j} = DS(A, d) - DS(B, d)$. Define $\mu_C = \mu \cdot \frac{d}{d-1}$, $\mu_X = 1 + \frac{\mu}{d-1}$.

We define the residual function $Res : \mathbb{N} \cup \{0\} \mapsto \mathbb{R}$ as follows:

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

Accordingly, we define function $C : \mathbb{N} \cup \{0\} \mapsto \mathbb{Z}$ as follows:

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

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

We define, $CSum(i) = \Sigma_{j=0}^{i} C(i) \cdot \frac{1}{d^j}$.

Lastly, we define $X : \mathbb{N} \cup \{0\} \mapsto \mathbb{R}$ s.t $X(i) = (DS^-(A, B, d, i) - CSum(i)) \cdot d^i$.

Lemma 7. *For all $i \geq 0$, $Res(i) = DS^-(A, B, d, \cdot) - CSum(i)$.*

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

Lemma 8. *When $DS^-(A, B, d, \cdot) \geq 0$, for all $i \geq 0$, $0 \leq Res(i) < \frac{1}{d^i}$.*

Proof. Since, $DS^-(A, B, d, \cdot) \geq 0$, $Res(0) = DS^-(A, B, d, \cdot) - \lfloor DS^-(A, B, d, \cdot) \rfloor \geq 0$ and $Res(0) = DS^-(A, B, d, \cdot) - \lfloor DS^-(A, B, d, \cdot) \rfloor < 1$. 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 9. *When $DS^-(A, B, d, \cdot) \geq 0$, for $i = 0$, $0 \leq C(0) \leq \mu_C$, and for $i \geq 1$, $0 \leq C(i) < d$.*

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

From Lemma 8, we know that for all i, $0 \leq Res(i) < \frac{1}{d^i}$. Alternately, when $i \geq 1$, $0 \leq Res(i-1) < \frac{1}{d^{i-1}} \implies 0 \leq Res(i-1) \cdot d^i < \frac{1}{d^{i-1}} \cdot d^i \implies 0 \leq Res(i-1) \cdot d^i < d \implies 0 \leq \lfloor Res(i-1) \cdot d^i \rfloor < d \implies 0 \leq C(i) < d$.

Corollary 2. *When $DS^-(A, B, d, \cdot) \geq 0$, $Range(C)$ is finite.*

Proof. From definition of C, we know that $C(0)$ takes integral values only. Further, from Lemma 9, we know that $C(0)$ is bounded. Hence, $C(0)$ takes finitely many values.

For $i > 0$, from definiton of C it is clear that $C(i)$ is an integer. Lemma 9 shows that each $C(i)$ is bounded by a fixed constant. Hence there are finitely many values of $C(i)$ for all other values of i.

Together, the above two show that $Range(C)$ is finite.

Lemma 10. *When $DS^-(A,B,d,\cdot) \geq 0$, then for all $i \geq 0$, $|X(i)| \leq \mu_X$.*

Proof. From definition of X, we know that $X(i) = (DS^-(A,B,d,i) - CSum(i)) \cdot d^i \implies X(i) \cdot \frac{1}{d^i} = DS^-(A,B,d,i) - CSum(i)$. From Lemma 7 we get $X(i) \cdot \frac{1}{d^i} = DS^-(A,B,d,i) - (DS^-(A,B,d,\cdot) - Res(i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (DS^-(A,B,d,\cdot) - DS^-(A,B,d,i)) \implies X(i) \cdot \frac{1}{d^i} = Res(i) - (\Sigma_{j=i+1}^{\infty}(A[j] - B[j]) \cdot \frac{1}{d^j}) \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + |(\Sigma_{j=i+1}^{\infty}(A[j] - B[j]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |(\Sigma_{j=0}^{\infty}(A[j+i+1] - B[j+i+1]) \cdot \frac{1}{d^j})| \implies |X(i) \cdot \frac{1}{d^i}| \leq |Res(i)| + \frac{1}{d^{i+1}} \cdot |\mu_C|$. From Lemma 8, this implies $|X(i) \cdot \frac{1}{d^i}| \leq \frac{1}{d^i} + \frac{1}{d^{i+1}} \cdot |\mu_C| \implies |X(i)| \leq 1 + \frac{1}{d} \cdot |\mu_C| \implies |X(i)| \leq 1 + \frac{\mu}{d-1} \implies |X(i)| \leq \mu_X$

Corollary 3. *When $DS^-(A,B,d,\cdot) \geq 0$, $Range(X)$ is finite.*

Proof. From expanding $X(i)$, we see that each $X(i)$ is an integer. Since $X(i)$ is bounded (see Lemma 10), this proves that $Range(X)$ is finite.

We overload notation, and define number sequences C and X s.t. for all $i \geq 0$, $C[i] = C(i)$ and $X[i] = X(i)$.

Lemma 11. *When $DS^-(A,B,d,\cdot) \geq 0$, then A, B, C and X satisfy the following invariant*

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

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

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

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

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

We construct state sequence $S = \{s\}_i$ as follows: Suppose $j \geq 0$ is the first instance where $C[j] > 0$

$$s_i = \begin{cases} s & \text{if } i = 0 \\ (X[i-1], \perp) & \text{if } 0 < i \leq j \\ (X[i-1], C[i-1]) & \text{if } i > j \end{cases}$$

Lemma 12. *When $DS^-(A,B,d,\cdot) \geq 0$, then S is a valid run of (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$.*

Proof. This is straight forward from Lemma 11 and construction of $\mathcal{A}_{\succ_{DS(d)}}$. Note that the construction makes use of finiteness of $Range(C)$ (See Corollary 2) and $Range(X)$ (See Corollary 3)

Corollary 4. *Let A and B be two non-zero bounded (by μ) integer sequences. Suppose $DS(A,d) > DS(B,d)$, then (A,B) has an accepting run in $\mathcal{A}_{\succ_{DS(d)}}$.*

Proof. For the given sequences A and B, generate sequences C and X as defined by functions C and X respectively. Then from Lemma 4 and Lemma 11, we know that $DS(C,d) > 0$. Since $0 \leq C[i]$, there exists at least on i where $C[i] \neq 0$. Let j be the first such index. Then from S we see that the state run moves into states in $\mathcal{F}$ in the j-th transition, and remains in $\mathcal{F}$ thereon. Therefore this run an accepting run for (A,B) in $\mathcal{A}_{\succ_{DS(d)}}$.

Theorem 2. *Let d be a positive integer, and $\mu \in \mathbb{Z}^+$. There exists a Büchi automaton $\mathcal{A}_{\succ_{DS(d)}}$ such that $(A,B) \in \mathcal{A}_{\succ_{DS(d)}}$ if and only if $DS(A,d) > DS(B,d)$ for all non-negative integer sequences A and B bounded by μ.*

Proof. Immediate from Corollary 1 and Corollary 4.

D Limit Average Comparator

Lemma 13. *Let $\Sigma = \{0, 1 \dots \mu\}$. Let $L \in \Sigma^*$ s.t. the limit-average of all words in the language L^ω exists. Then average of all words in L is the same.*

Proof. Suppose it is possible that two finite words v_1, $v_2 \in L$ have different average. Let their length be l_1 and l_2 respectively with the average a_1 and a_2 respectively where $a_1 \neq a_2$. We will show the presence of a word $w \in L^\omega$ s.t. the limit-average of w does not exist.

Let $w_1 = v_1$. Then $\mathsf{Avg}(w_1) = a_1$. Next, let j_2 be large enough to construct $w_2 = w_i v_2^{j_2}$ such that $\mathsf{Avg}(w_2) \approx a_2$. Next, let j_3 be large enough to construct $w_3 = w_2 v_1^{j_3}$ such that $\mathsf{Avg}(w_3) \approx a_1$. Continue constructing $w_4, w_5 \dots$ in a similar fashoin s.t. their average change between $a_2, a_1 \dots$ respectively.

Let these $w = w_n$ as $n \to \infty$. Then $w \in L^\omega$, and since the average of its finite-length prefixes keeps changing between a_1 and a_2, limit-average of w does not exist.

This contradicts the premise that the limit-average of all words in L^ω exists. Therefore, our assumption that words v_1 and v_2 can have different average has been contradicted.

Lemma 14. *Let $\Sigma = \{0, 1, \dots \mu\}$. Let $L \subseteq \Sigma^*$ s.t. the average of all words in Σ is the same, say a.*

Let $w \in L^\omega$ s.t. limit-average of w exists. Then $\mathsf{LimAvg}(w) = a$

Proof. Let $w = w_1 w_2 w_3 \dots$. There exists infinitely many prefixes of w, prefix $w[i] = w_1 w_2 \dots w_i$ s.t. $\mathsf{Avg}(w[i]) = a$.

Since we are given that the limit-average of w exists, it must be equal to a.

Lemma 15. *Let $\Sigma = \{0, 1 \ldots, \mu\}$. Let $\mathcal{L}_{LA} \subseteq \Sigma^\omega$.*
$\mathcal{L}_{LA}$ is not an ω-regular language.

Proof. 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. From Lemma 13, we conclude that the average of all word in V_i must be the same. Furthermore, from Lemma 14, we know that 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.

Lemma 16. *Let $\Sigma = \{0, 1 \ldots, \mu\}$. Let $\mathcal{L}_{LA} \subseteq \Sigma^\omega$.*
$\mathcal{L}_{LA}$ is not an ω-context-free language.

Proof. 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 that of Lemma 15.

Lemma 17. *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. 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 18. *Let A and B be sequences s.t. their limit average exists. If $\exists^\infty i, \mathsf{Sum}(A[0, i-1]) > \mathsf{Sum}(B[0, i-1])$ (or $\exists^\infty i, \mathsf{Sum}(A[0, i-1] \geq \mathsf{Sum}(B[0, i-1]))$) then $\mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$.*

Proof. 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 17, 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 $\mathsf{LimAvg}(A) \geq \mathsf{LimAvg}(B)$.

Lemma 19. *For sequences A and B, if $\mathsf{LimSupAvg}(A) > \mathsf{LimSupAvg}(B) > \mathsf{LimInfAvg}(B) > \mathsf{LimInfAvg}(A)$ then A and B exhibit oscillatory behavior.*

Proof. Let $\mathsf{LimInfAvg}(A)$, $\mathsf{LimSupAvg}(A)$, $\mathsf{LimInfAvg}(B)$, and $\mathsf{LimSupAvg}(B)$ be denoted by a_i, a_s, b_i, and b_s respectively.

For all $\epsilon > 0$ there exists an N_ϵ s.t. $|\frac{\mathsf{Sum}(A[1,N_\epsilon])}{N_\epsilon} - a_s| < \epsilon$, and $\frac{\mathsf{Sum}(B[1,N_\epsilon])}{N_\epsilon} < b_s + \epsilon$. Let $a_s - b_s = k_s > 0$. Take $\epsilon = \frac{k_s}{2}$. Then $\mathsf{Sum}(A[1,N_{k_s}]) > \mathsf{Sum}(B[1,N_{k_s}]) > 0$. Arguing similarly for $\mathsf{LimInfAvg}(A)$ and $\mathsf{LimInfAvg}(B)$, there exists N_{k_i} where $\mathsf{Sum}(B[1,N_{k_i}]) > \mathsf{Sum}(A[1,N_{k_i}]) > 0$.

Let $N = max\{N_{k_s}, N_{k_i}\}$. Note that $A[N\ldots]$ and $B[N\ldots]$ have the same values for limit average infimum and limit average supremum. Repeated the earlier process to detect appropriate N_{k_s} and N_{k_i} as before, and truncate $A[N\ldots]$ and $B[N,\ldots]$ further.

This way one can obtain infinitely many indexes where $\mathsf{Sum}(A[1,i]) > \mathsf{Sum}(B[1,i]) > 0$ and infinitely many indexes where $\mathsf{Sum}(B[1,i]) > \mathsf{Sum}(A[1,i]) > 0$.

Hence, we have proven that A and B exhibit oscillatory behavior.

E Prefix Average Comparison

Lemma 20. *For sequences A and B, if $\mathsf{LimInfAvg}(A) > \mathsf{LimSupAvg}(B)$ then $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.*

Proof (Proof Sketch). Let $\mathsf{LimInfAvg}(A)$ and $\mathsf{LimSupAvg}(B)$ 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 17 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 $\mathsf{Sum}(A[0,i-1]) > \mathsf{Sum}(B[0,i-1])$

F Prefix Average Comparator

Construction We provide a sketch of the construction of the Büchi push-down autoamaton $\mathcal{A}_{\succeq_{PA(\cdot)}}$, 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 $\mathcal{A}_{\succeq_{PA(\cdot)}} = (S, \Sigma \times \Sigma, \Gamma, \delta, s_0, Z_0)$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$ accepts words by final state.
– Here we give a sketch of the behavior of the transition function δ.
  - When $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 21. *Push down automaton $\mathcal{A}_{\succeq_{PA(\cdot)}}$ accepts a pair of sequences (A, B) iff* $\mathsf{PrefixAvg}(A) \geq \mathsf{PrefixAvg}(B)$.

Proof (Proof sketch). To prove this statement, it is sufficient to demonstrate that $\mathcal{A}_{\succeq_{LA(\cdot)}}$ accepts a pair of sequences (A, B) iff there are only finitely many indexes where $\mathsf{Sum}(B[1, i]) > \mathsf{Sum}(A[1, i])$. On $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $m > k$, $\mathsf{Sum}(B[1, m]) \not\geq \mathsf{Sum}(A[1, m])$. If a run of (A, B) switches to F at this m, then it will be accepted by the push down automaton. Since $\mathcal{A}_{\succeq_{LA(\cdot)}}$ 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 $\mathcal{A}_{\succeq_{LA(\cdot)}}$.