Institute of Science and Technology

Nested Weighted Automata

Krishnendu Chatterjee and Thomas A. Henzinger and Jan Otop

Technical Report No. IST-2014-170-v1+1
Deposited at 19 Feb 2014 10:12
http://repository.ist.ac.at/170/1/main.pdf

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Nested Weighted Automata*

Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop**

IST Austria

Abstract. Recently there has been a significant effort to add quantitative properties in formal verification and synthesis. While weighted automata over finite and infinite words provide a natural and flexible framework to express quantitative properties, perhaps surprisingly, several basic system properties such as average response time cannot be expressed with weighted automata. In this work, we introduce nested weighted automata as a new formalism for expressing important quantitative properties such as average response time. We establish an almost complete decidability picture for the basic decision problems for nested weighted automata, and illustrate its applicability in several domains.

1 Introduction

Traditionally, formal verification has focused on Boolean properties of systems, such as "every request is eventually granted." Automata-theoretic formalisms as well as temporal logics have been studied as specification languages for such Boolean properties of reactive systems. In recent years there has been a growing trend to extend specifications with quantitative aspects for expressing properties such as "the long-run average success rate of an operation is at least one half" or "the long-run average (or the maximal, or the accumulated) resource consumption is below a threshold." Quantitative aspects of specifications are essential for resource-constrained systems, such as embedded systems, and for performance evaluation.

Weighted automata provide a natural and flexible framework for expressing quantitative[1] properties [8]. Weighted automata are an extension of finite automata in which every transition is labeled by a rational weight. Thus, each run produces a sequence of weights, and a value function compresses the sequence into a single value. For non-deterministic weighted automata, the value of a word w is the infimum value of all runs over w. Initially, weighted automata were studied over finite words with weights from a semiring and ring multiplication as value function [10]. They have been extended to infinite words with limit averaging or supremum as value function [8, 7, 5]. While weighted automata over semirings can express several quantitative properties [15], they cannot express long-run averages.

Yet even weighted automata with limit-average value function, perhaps surprisingly, are not capable of expressing basic quantitative properties of systems such as the long-run average response time. The long-run average response time is the limit average of all

* This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), by the Austrian Science Fund (FWF) project S11402-N23 (RiSE), FWF Grant No P23499- N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

** The third autor is on the leave from University of Wrocław

[1] We use the term "quantitative" in a non-probabilistic sense, which assigns a quantitative value to each infinite run of a system, representing long-run average or maximal response time, or power consumption, or the like, rather than taking a probabilistic average over different runs.

distances between requests and subsequent grants along an infinite run. To see this, notice that the value of a weighted automaton with limit-average value function is bounded by the maximal weight that occurs in the automaton, whereas the long-average response time can be unbounded. However, the long-run average response time can be computed if the sum value function can be applied between requests and subsequent grants, and the values of the sum function can be aggregated using a limit-average function. Such a mechanism can be expressed naturally by an extension of weighted automata, called *nested weighted automata*, which we introduce in this paper.

A nested weighted automaton consists of a master automaton and a set of slave automata. The master automaton runs over an infinite word, and at each transition of the infinite run, it may invoke a slave automaton that runs over a finite subword of the infinite word, starting from the position where the master automaton invokes the slave automaton. Each slave automaton terminates after a finite number of steps and returns a value to the master automaton. To compute its return value, each slave automaton is equipped with a value function for finite words, and the master automaton aggregates all return values using a value function for infinite words. While in case of unweighted finite automata, nested automata are no more expressive than their non-nested counterpart, the class of nested weighted automata is strictly more expressive than non-nested weighted automata. For example, with nested weighted automata, the long-run average response time of a word can be computed as follows: the master automaton has the limit-average value function, and at every request, it invokes a slave automaton with sum value function, which counts the number of transitions to the next grant.

Our contributions are three-fold. First, we introduce nested weighted automata over infinite words (Section 3), which is a new formalism for expressing important quantitative properties, such as long-run average response time, which cannot be specified by non-nested weighted automata.

Second, we study the decidability of emptiness, universality, and inclusion for nested weighted automata. We present an almost complete decidability picture for several natural and well-studied value functions. On the positive side, we show that if the value functions of the slave automata are max, min, or bounded sum, then the decision problems for nested weighted automata can be reduced to corresponding problems for non-nested weighted automata. Moreover, we show that if the value function of the master automaton is limit average and the value function of the slave automata is non-negative sum (i.e., sum over non-negative weights), which included the long-run average response time property, then the emptiness question is decidable. The decidability proof is obtained by establishing certain regularity properties of optimal runs, which can be used to reduce the problem to the emptiness question for a non-nested weighted automata with limit-average value function. On the negative side, we show that even for deterministic nested weighted automata with sup value function for the master automaton and sum value function for the slave automata, the emptiness question is undecidable. This result is in sharp contrast to non-nested weighted automata, where the emptiness and universality questions are always decidable for deterministic automata, and the emptiness question is decidable also for non-deterministic sup and sum automata. Our results are summarized in Table 1 and Table 2.

Third, we illustrate the applicability of nested weighted automata in several domains. (1) We show that nested weighted automata provide a convenient and expressive formalism to specify quantitative properties of systems such as long-run average response time (Example 10 in Section 5.1). Nested weighted automata can be seen a quantitative extension of monitor automata for the verification of Boolean properties [16], which are used heavily also in run-time verification [13]. In case of Boolean properties, each monitor automaton

tracks a subproperty (which corresponds to slave automata in our formalism), and the results of the monitor automata are combined by a tester automaton (which corresponds to the master automaton in our formalism). Hence our framework can also be seen as a first step towards quantitative run-time verification, where the slave automata return values of subproperties, and the master automaton (assuming a commutative value function) computes an on-the-fly the approximation of the value. (2) We show that the *model-measuring* problem of [14] can be expressed in the nested weighted automaton framework (Section 5.2). The model-measuring problem asks, given a model and a specification, how robustly the model satisfies the specification, i.e., how much the model can be perturbed without violating the specification. (3) As dual of the model-measuring problem, we introduce the *model-repair* problem and show that it, as well, can be solved using nested weighted automata (Section 5.3). The model-repair problem asks, given a specification and a model that does not satisfy the specification, for the minimal restriction of the model so that it satisfies the specification. We show that we need nested weighted automata in order to express interesting measures on models for the model-measuring and model-repair problems.

Related work. Weighted nested automata have been considered in [3] in the context of finite words, where the weights are given oven semirings (it was further required that the semirings of all master and slave automata coincide, while in our case, their value functions may differ). The main objective of [3] was to assign values to Boolean nested automata, whereas we consider nesting of weighted automata to express properties of infinite behaviors of systems. Properties such as long-run average response time cannot be expressed in the framework of [3]. Deterministic automata with registers have been studied in [2] over finite words. Our work is different because we consider infinite words and the nested control of automata. For example, the emptiness of register automata with max and sum value functions is decidable, while we show it is undecidable for deterministic nested weighted automata with these value functions.

2 Preliminaries

Words. Given a finite alphabet Σ of letters, a finite (resp. infinite) word w is a finite (resp. infinite) sequence of letters. For a word w, we define $w[i]$ as the i-th letter of w and $w[i,j]$ as the word $w[i]w[i+1]\dots w[j]$. We allow j to be ∞ for infinite words. For a finite word w, we denote by $|w|$ its length; and for an infinite word the length is ∞.

Non-deterministic automata. A *(non-deterministic) automaton* $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F)$, where Σ is the alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of *accepting* states.

Runs. Given an automaton $\mathcal{A}$ and a word w, a *run* $\pi = \pi[0]\pi[1]\dots$ is a sequence of states such that $\pi[0] \in Q_0$ and for every $i \in \{1, \dots, |w|\}$ we have $(\pi[i-1], w[i], \pi[i]) \in \delta$. Given a word w, we denote by $\mathsf{Run}(w)$ the set of all possible runs on w.

Boolean acceptance. The acceptance of words is defined using the accepting states. A finite run π of length j is *accepting* if $\pi[j] \in F$; and an infinite run π is *accepting*, if there exists infinitely many j such that $\pi[j] \in F$. Let $\mathsf{Acc}(w) \subseteq \mathsf{Run}(w)$ denote the set of accepting runs, and a word w is accepted, if $\mathsf{Acc}(w)$ is non-empty. We denote by $\mathcal{L}_{\mathcal{A}}$ the set of words accepted by $\mathcal{A}$.

Labeled and weighted automata. Given a finite alphabet Γ, a *Γ-labeled automaton* is an automaton whose transitions are labeled by elements from Γ. Formally, a labeled automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, Q_0, \delta, F, C)$ such that $(\Sigma, Q, Q_0, \delta, F)$ is an automaton and $C : \delta \mapsto \Gamma$.

A *weighted* automaton is a labeled automaton where $\Gamma \subseteq \mathbb{Q}$, where $\mathbb{Q}$ is the set of rationals; and the labels of the transitions are referred to as *weights*.

Semantics of weighted automata. To define the semantics of weighted automata we need to define the value of a run (that combines the sequence of weights of a run to a single value) and the value across runs (that combines values of different runs to a single value). To define values of runs, we will consider *value functions* f that assign real numbers to sequences of rationals. Given a word w, every run π of $\mathcal{A}$ on w defines a sequence of weights of successive transitions of $\mathcal{A}$, i.e., $C(\pi) = (C(\pi[i-1], w[i], \pi[i]))_{1 \leq i \leq |w|}$; and the value of the run π is defined as $f(C(\pi))$. We will denote by $(C(\pi))[i]$ the cost of the i-th transition, i.e., $C(\pi[i-1], w[i], \pi[i])$. The value of a word w assigned by the automaton $\mathcal{A}$, denoted by $\mathcal{L}_{\mathcal{A}}(w)$, is the infimum of the set of values of all *accepting* runs; i.e., $\inf_{\pi \in \mathsf{Acc}(w)} f(\pi)$, and we have the usual semantics that infimum of an empty set is infinite. To indicate a particular value function f that defines the semantics, we will call a weighted automaton $\mathcal{A}$ an f-automaton.

Types of automata. A weighted automaton is

- *deterministic* iff Q_0 is singleton and the transition relation is a function; and
- *functional* iff for every word w, all accepting runs on w have the same value.

Value functions. We will consider the classical functions and their variants for value functions. For finite runs we consider the following value functions: for runs of length n we have

1. *Max and min:* $\textsc{Max}(\pi) = \max_{i=1}^{n}(C(\pi))[i]$ and $\textsc{Min}(\pi) = \min_{i=1}^{n}(C(\pi))[i]$.
2. *Sum and variants:* the sum function $\textsc{Sum}(\pi) = \sum_{i=1}^{|\pi|}(C(\pi))[i]$, the absolute sum $\textsc{Sum}^{+}(\pi) = \sum_{i=1}^{|\pi|} \mathsf{Abs}((C(\pi))[i])$ is sum of the absolute values of the weights (Abs denotes the absolute value of a number), and the bounded sum objective returns the sum if all the partial absolute sums are below a bound B, otherwise it returns the bound B, i.e., formally, $\textsc{Sum}^{B}(\pi) = \textsc{Sum}(\pi)$, if for all prefixes π' of π we have $\mathsf{Abs}(\textsc{Sum}(\pi')) \leq B$, otherwise B.

We denote the above class of value functions for finite words as $\mathsf{FinVal} = \{\textsc{Max}, \textsc{Min}, \textsc{Sum}, \textsc{Sum}^{+}, \textsc{Sum}^{B}\}$. For infinite runs we consider:

1. *Supremum and Infimum, and Limit supremum and Limit infimum*: $\textsc{Sup}(\pi) = \sup\{(C(\pi))[i] : i > 0\}$, $\textsc{Inf}(\pi) = \inf\{(C(\pi))[i] : i > 0\}$, $\textsc{LimSup}(\pi) = \limsup\{(C(\pi))[i] : i > 0\}$, and $\textsc{LimInf}(\pi) = \liminf\{(C(\pi))[i] : i > 0\}$.
2. *Limit average:* $\textsc{LimAvg}(\pi) = \limsup_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k}(C(\pi))[i]$.

We denote the above class of value functions for infinite words as $\mathsf{InfVal} = \{\textsc{Sup}, \textsc{Inf}, \textsc{LimSup}, \textsc{LimInf}, \textsc{LimAvg}\}$.

Decision questions. We consider the standard emptiness and universality questions. Given an f-automaton $\mathcal{A}$ and a threshold λ, the *emptiness* (resp. *universality*) question asks whether there exists a word w such that $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$ (resp. for all words w we have $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$). We summarize the main results from the literature related to the decision questions of weighted automata for the class of value functions defined above.

Theorem 1. *(1) The emptiness problem is decidable for all value functions we consider [11, 15]. (2) The universality problem is undecidable for* SUM*-automata with* $\{-1, 0, 1\}$ *weights and* LIMAVG *automata with* $\{0, 1\}$ *weights; and decidable for all other value functions [1, 9, 4]. (3) The universality problem is decidable for all value functions for deterministic and functional automata [12].*

3 Nested Weighted Automata

In this section we introduce nested weighted automata. We start with an informal description.

Informal description. A *nested weighted automaton* consists of a labeled automaton over infinite words, called the *master automaton*, a value function f, and a set of weighted automata over finite words, called *slave automata*. A nested weighted automaton can be viewed as follows: given an infinite word, we consider the run of the master automaton on the word, but the weight of each transition is determined by dynamically running slave automata; and then the value of a run is obtained using the value function f. That is, the master automaton proceeds on an input word as an usual automaton, except that before it takes a transition, it can start a slave automaton corresponding to the label of the current transition. The slave automaton starts at the current position of the word of the master automaton and works on some finite part of the input word. Once a slave automaton finishes, it returns its value to the master automaton, which treats the returned value as the weight of the current transition that is being executed. Note that for some transitions the master automaton might not invoke any slave automaton, and we refer to such transitions as *silent* transitions. The value of the run of the master automaton is given by f applied to the sequence of values returned by slave automata (i.e., to compute the value function the silent transitions are omitted). If one of the slave automata rejects, the nested weighted automaton rejects. We now formalize the informal description.

Nested weighted automata. A *nested weighted automaton* is a tuple $\mathbb{A} = \langle \mathcal{A}_{mas}; f; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$, where $\mathcal{A}_{mas}$ is an automaton over infinite words labeled by $\{1, \ldots, k\} \cup \{\bot\}$ either numbers (corresponding to indices of slave automata) or $\bot$ (for silent transitions), called the *master* automaton, f is a value function on infinite sequences, and $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$ are weighted automata over finite words, called *slave* automata.

Semantics: runs and values. Let w be an infinite word. A *run* of $\mathbb{A}$ on w is an infinite sequence $(\Pi, \pi_1, \pi_2, \ldots)$ such that (i) Π is a run of $\mathcal{A}_{mas}$ on w; (ii) idx obtained as the sub-sequence of labels of transitions taken in Π by removing all $\bot$-labeled transitions is an infinite sequence; and (iii) for every $i > 0$ we have π_i is a run of the automaton $\mathfrak{B}_{\mathsf{idx}[i]}$, referenced by the label of the master automaton, on some finite word of $w[pos(i), j]$, where $pos(i)$ denotes the position of the word where the i-th non-silent transition is taken and $j \geq pos(i)$. The run $(\Pi, \pi_1, \pi_2, \ldots)$ is accepting if all runs $\Pi, \pi_1, \pi_2, \ldots$ are accepting (i.e., Π satisfies its acceptance condition and each $\pi_1, \pi_2, \ldots$ ends in an accepting state). The value of the run $(\Pi, \pi_1, \pi_2, \ldots)$ is defined as $f(v(\pi_1)v(\pi_2)\ldots)$, where $v(\pi_i)$ is the value of the run π_i in the corresponding slave automaton. The value of a word w assigned by the automaton $\mathbb{A}$, denoted by $\mathcal{L}_{\mathbb{A}}(w)$, is the infimum of the set of values of all *accepting* runs.

Notation. Let f, g be value functions. We say that a nested weighted automaton $\mathbb{A} = \langle \mathcal{A}_{mas}; h; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$ is an $(f; g)$-automaton iff $h = f$ and $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$ are g-automata (weighted automata over finite words with value function g). We illustrate the semantics of nested weighted automata with an example.

Example. Consider the nested weighted automaton $\mathbb{A}_{stu} = \langle \mathcal{A}^1_{mas}; \textsc{LimAvg}; \mathfrak{B}_1, \mathfrak{B}_2 \rangle$ where each slave automaton is a $\textsc{Sum}^+$-automaton. The automaton $\mathcal{A}^1_{mas}$ has a single state and two transitions (q_0, a, q_0) labeled by 1 and (q_0, b, q_0) labeled by 2. The slave automaton $\mathfrak{B}_1$ runs as long as it sees a letters, increasing its value by 1 at each step; and once it sees b, it terminates. In other words, it counts the number of a letters to the first occurrence of b. The automaton $\mathfrak{B}_2$ is virtually the same, but it counts b letters until it sees a. Consider a

word $(aaab)^\omega$ and its run is depicted in Figure 1. The value of the word is $\frac{3}{4}$. Note that $\mathbb{A}_{stu}$ accepts only words with infinite number of a's and b's, as otherwise, some slave automaton would have to run infinitely long and not terminate. For word $w = (a^n b)^\omega$ the value is $\frac{(n-1)n}{2(n+1)}$, and this shows that the nested weighted automaton can return unbounded values (in contrast to a LIMAVG-automaton whose range is bounded by its maximal weight). Consider a variant of the above automaton where we change the master automaton to $\mathcal{A}^2_{mas}$ shown in Fig 1. Intuitively, the slave automata counts the length of each block of a's and b's and thus the value computed by the nested automaton is the average block length (or average stuttering).

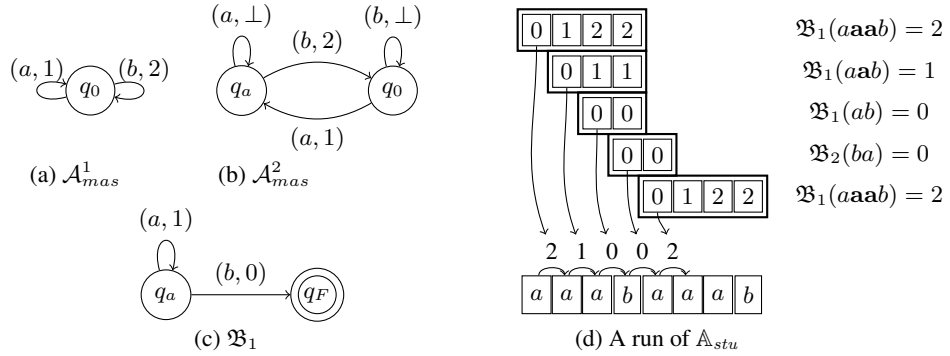


Fig. 1: The master automata (a) $\mathcal{A}^1_{mas}$, (b) $\mathcal{A}^2_{mas}$, the slave automaton (c) $\mathfrak{B}_1$ and a run of the nested automaton $\mathbb{A}_{stu}$.

Equivalence with weighted automata. We say that a nested weighted automaton $\mathbb{A}$ and a weighted automaton $\mathcal{A}$ are *equivalent* iff their values coincide on each word, i.e., for all $w \in \Sigma^\omega$ we have $\mathcal{L}_{\mathbb{A}}(w) = \mathcal{L}_{\mathcal{A}}(w)$.

Determinism of nested weighted automata. There are two reasons why a nested automaton may be non-deterministic. The first one is standard: one of the components, the master automaton or one of the slave automata is non-deterministic. The second one is more subtle: it is the termination of slave automata. To accept, a slave automaton has to terminate in an accepting state, but it not need to be the first time it visits an accepting state. It can run longer to compute a different value. However, if the language $\mathcal{L}$ recognized by the slave automaton is *prefix-free*, i.e., $w \in \mathcal{L}$ implies that no extension of w belongs to $\mathcal{L}$, then it has to terminate once it reaches an accepting state because it will have no other chance to accept. This intuition suggests the following definition.

Types of nested weighted automata. A nested weighted automaton is *deterministic* iff the master automaton and all slave automata are deterministic and each slave automaton recognizes a prefix-free language. A nested weighted automaton is *functional* iff for every word w, each accepting run on w has the same weight.

We will consider the decision questions of emptiness and universality for nested weighted automata.

4 Decision Problems

In this section we study the decidability of the decision problems for nested weighted automata. We start with some simple observations.

Simple observations. Note that the emptiness (resp. universality) of f-automata and g-automata reduces to the emptiness (resp. universality) of $(f; g)$-automata: by simply considering dummy master or dummy slave automata. Hence by Theorem 1 it follows that the universality problem is undecidable for $(f; \textsc{Sum})$-automata and $(\textsc{LimAvg}; g)$-automata, where $f \in \mathsf{InfVal}$ and $g \in \mathsf{FinVal}$.

Theorem 2. *(1) For $f \in \mathsf{InfVal}$, the universality problem for $(f; \textsc{Sum})$-automata is undecidable. (2) For $g \in \mathsf{FinVal}$, the universality problem $(\textsc{LimAvg}; g)$-automata is undecidable.*

4.1 Regular Weighted Slave Automata

We present a general result that ensures decidability for the decision problems for a large class of nested weighted automata. We now consider slave automata that can only return values from a bounded domain, and present decidability results for them.

Definition 3 (Regular weighted automata). *Let $\mathcal{A}$ be a weighted automaton over finite words. We say that the weighted automaton $\mathcal{A}$ is a* regular weighted *automaton iff there is a finite set $\{q_1, \ldots, q_n\} \subseteq \mathbb{Q}$ and there are regular languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$ such that*

(i) every word accepted by $\mathcal{A}$ belongs to $\bigcup_{1 \leq i \leq n} \mathcal{L}_i$, and
(ii) for every $w \in \mathcal{L}_i$, each run of $\mathcal{A}$ on w has the weight q_i.

Regular value functions. A value function f is a *regular value function* iff all f-automata are regular weighted automata. Examples of regular value functions are $\textsc{Min}, \textsc{Max}, \textsc{Sum}^B$.

Key reduction lemma. In the following key lemma we establish that if the slave automata are regular weighted automata, then nested weighted automata can be reduced to weighted automata with the same value function as for the master automata. For regular weighted slave automata, a weighted automaton can simulate a nested automaton in the following way. Instead of starting a slave automaton, the weighted automaton guesses the weight of the current transition (i.e., the value to be returned of the slave automaton) and checks that the guessed weight is correct. The definition of regular weighted automata implies that such a check can be done by a (non-weighted) finite automaton $\mathcal{S}$. Thus, the weighted automaton takes a universal transition such that in one branch it continues its execution and in another it runs $\mathcal{S}$. Observe that such a universal transition can be removed by a standard power-set construction. Given a value function f, we denote by $\mathsf{sil}(f)$ the value function that applies f on sequences after removing silent transitions. Lemma 4 along with Theorem 1 implies Theorem 5.

Lemma 4 (Key reduction lemma). *Let $f \in \mathsf{InfVal}$ be a value function. Consider a nested weighted automaton $\mathbb{A} = \langle \mathcal{A}_{mas}; f; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$ such that all automata $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$ are regular weighted automata. There is an $\mathsf{sil}(f)$-automaton $\mathcal{A}$ (weighted automaton), that can be constructed in exponential time, which is equivalent to $\mathbb{A}$; moreover, if $\mathbb{A}$ is functional, then $\mathcal{A}$ is functional as well.*

Theorem 5. *Let $g \in \{\textsc{Min}, \textsc{Max}, \textsc{Sum}^B\}$. The following assertions hold: (1) The emptiness and universality problems are decidable for non-deterministic $(f; g)$-automata, where $f \in \{\textsc{Inf}, \textsc{Sup}, \textsc{LimInf}, \textsc{LimSup}\}$. (2) The emptiness problem is decidable for non-deterministic $(\textsc{LimAvg}; g)$-automata. (3) The universality problem is decidable for functional $(\textsc{LimAvg}; g)$-automata.*

Theorem 5 covers the case for all classes of slave automata other than SUM- and SUM$^+$-automata, which we consider in the following two subsections.

4.2 Undecidability Results for Slave SUM Automata

In this section we study $(f; \text{SUM})$-automata and $(f; \text{SUM}^+)$-automata. In contrast to the reduction of Lemma 4, for example, $(\text{LIMAVG}; \text{SUM}^+)$ cannot be reduced to weighted LIMAVG automata (Example 10).

Crucial negative result. We now present a crucial negative result. Note that for weighted automata the emptiness problem is always decidable (for non-deterministic automata); and all decision problems are decidable for deterministic automata. In sharp contrast we establish that for deterministic $(\text{SUP}; \text{SUM})$ automata the emptiness problem is undecidable. The proof is a reduction from the halting problem of a two-counter (Minsky) machine to the emptiness problem. The key idea is to ensure that words that encode valid computations of the Minsky machine have value 0; and all invalid computations have value strictly greater than 0. Basically, we need to check consistency of values of each counter at each step, which is done as follows. The task of the master automaton is to ensure that tests on the counters are consistent. The master automaton uses several slave automata to track the exact values of the counters. Each slave automaton operates on an alphabet which is increment and decrement for the counters, as well as zero and positive test, and for each counter we have three slave automata. For positions $i < j$, let *balance* between position i and j denote the difference in the number of increments and decrements between i and j. For zero tests of a counter, two slave automata are invoked: the first automaton (resp. second automaton) increments (resp. decrements) with every increment operation on the counter and decrements (resp. increments) with every decrement operation on the counter and terminates with the value at the position of the next zero test. Intuitively, the two automata compute the balance and negative of the balance between two consecutive zero tests. Given the zero test of the current position is satisfied, both automata returns zero iff the next zero test is also satisfied, otherwise one of them return a positive value. For positive tests of a counter we use the third slave automaton to compute the balance plus 1 between the current position and the next zero test. The balance plus 1 do not exceed zero iff the current counter value is positive. This establishes the undecidability for emptiness of $(\text{SUP}; \text{SUM})$-automata, and the proof also holds for $(\text{LIMSUP}; \text{SUM})$-automata. Also observe that since we establish the result for deterministic automata, we can take inverses of weights and change SUP (resp. LIMSUP) to INF (resp. LIMINF) and the emptiness problem to the universality problem.

Theorem 6 (Crucial undecidability result). *(1) The emptiness problem for deterministic* $(\text{SUP}; \text{SUM})$- *and* $(\text{LIMSUP}; \text{SUM})$*-automata is undecidable. (2) The universality problem for deterministic* $(\text{INF}; \text{SUM})$- *and* $(\text{LIMINF}; \text{SUM})$*-automata is undecidable.*

4.3 Decidability Results for Slave SUM- and SUM$^+$-Automata

We now establish the remaining decidability results, namely, for slave automata with SUM$^+$ value functions, and emptiness for $(\text{INF}; \text{SUM})$-automata and $(\text{LIMINF}; \text{SUM})$-automata.

Intuitive proof ideas. For $(f; \text{SUM}^+)$-automata, for $f \in \mathsf{InfVal} \setminus \{\text{LIMAVG}\}$, we show that the decision problems can be reduced to the bounded sum value function; and then derive the decidability results from Theorem 5. For $(\text{INF}; \text{SUM})$-automata we show the emptiness problem is decidable and the main argument is a reduction to the emptiness of non-deterministic weighted automata with SUM value function. We summarize the results in the following theorem.

Theorem 7. *(1) The emptiness problem for* $(\textsc{Inf}; \textsc{Sum})$*-automata and* $(\textsc{LimInf}; \textsc{Sum})$*-automata is decidable. (2) The universality problem for functional* $(\textsc{Sup}; \textsc{Sum})$*-automata and* $(\textsc{LimSup}; \textsc{Sum})$*-automata is decidable. (3) For* $f \in \{\textsc{Inf}, \textsc{Sup}, \textsc{LimInf}, \textsc{LimSup}\}$*, the emptiness and the universality problems for* $(f; \textsc{Sum}^+)$*-automata are decidable.*

Finally, we establish decidability of the emptiness problem with limit-average master automaton and $\textsc{Sum}^+$-automata as slaves. The key proof idea is to show that there exist optimal runs for $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata that coincide with the values of optimal runs of a non-nested limit-average automata. This also allows us to show the decidability of the universality problem for functional $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata.

Theorem 8. *The emptiness problem for* $(\textsc{LimAvg}; \textsc{Sum}^+)$*-automata is decidable; and the universality problem for functional* $(\textsc{LimAvg}; \textsc{Sum}^+)$*-automata is decidable.*

While we have established the decidability and undecidability of the decision problems for nested weighted automata for almost all cases, there is one open problem which present as a conjecture. Tables 1, 2 summarize our results.

Conjecture 9. The emptiness problem for non-deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata is decidable.

		Inf	Sup	LimInf	LimSup	LimAvg
Min, Max, $\textsc{Sum}^B$	Emptiness	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)
	Universality	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)
Sum	Emptiness	Dec.(7)	Undec.(6)	Dec.(7)	Undec.(6)	Open (9)
	Universality	Undec.(6)	Dec.(7)	Undec.(6)	Dec.(7)	Open (9)
$\textsc{Sum}^+$	Emptiness	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(8)
	Universality	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(8)

Table 1: Decidability of the emptiness and universality problems for functional $(f; g)$-automata. Functions f are listed in the first row and functions g are in the first column. The undecidability results hold even for deterministic automata. Next to each result there is a reference to the corresponding theorem or conjecture.

		Inf	Sup	LimInf	LimSup	LimAvg
Min, Max, $\textsc{Sum}^B$	Emptiness	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)
	Universality	Dec.(4)	Dec.(4)	Dec.(4)	Dec.(4)	Undec.(2)
Sum	Emptiness	Dec.(7)	Undec.(6)	Dec.(7)	Undec.(6)	Open (9)
	Universality	Undec.(6)	Undec.(2)	Undec.(6)	Undec.(2)	Undec.(2)
$\textsc{Sum}^+$	Emptiness	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(8)
	Universality	Dec.(7)	Dec.(7)	Dec.(7)	Dec.(7)	Undec.(2)

Table 2: Decidability of the emptiness and universality problems for non-deterministic $(f; g)$-automata. The alignment is as in Table 1.

Discussion on expressiveness and inclusion. We discuss the expressiveness and inclusion results for nested weighted automata. (1) Nested weighted automata can be strictly more

expressive than their non-nested counterpart; for example, (LIMAVG; SUM$^+$) automata can express average response time property which cannot be expressed by LIMAVG-automata or SUM$^+$-automata (see Example 10 in Section 5.1). (2) The emptiness and universality problems reduce to the inclusion problem, where the inclusion problem given two automata $\mathbb{A}_1$ and $\mathbb{A}_2$ asks whether for every word w we have $\mathcal{L}_{\mathbb{A}_1}(w) \leq \mathcal{L}_{\mathbb{A}_1}(w)$. Therefore, for decidability of the inclusion problem both the emptiness and the universality problems must be decidable. We establish that the inclusion problem is decidable for $(f; g)$-automata where $f \in \mathsf{InfVal} \setminus \{\text{LIMAVG}\}$ and g is a regular function. For further discussion see Section G of Appendix.

5 Applications

In this section we discuss several applications of nested weighted automata.

5.1 Quantitative system properties

We first show that basic properties such as average response time can be expressed conveniently as a nested weighted automaton. We then argue that our framework is a natural extension of the framework of monitor automata for Boolean verification, and is a step towards quantitative run-time verification.

Example 10 (Average response time). Consider the specification for average response time defined as follows: we consider words for the alphabet $\{r, g, n\}$, where r denotes a request, g denotes a grant, and n denotes null (no request or grant). Consider a word w, and a position i, such that $w[i]$ is a request, and then the response time in position i is the distance to the closest grant, i.e., the response is $j - i$ where $j > i$ is the least number greater than i with $w[j] = g$. The average response time is the limit-average of the response times of the requests. Consider a nested weighted automaton, with one slave automaton that has sum of non-negative weights as the value function, and the master automaton with limit-average value function. The master automaton for every letter r invokes the slave automaton, and for g and n takes a silent transition (i.e., it is a single state automaton with r labeled as 1, and g and n labeled as $\perp$). The slave automaton counts the number of steps till the first g. The nested automaton specifies the average response time property. As discussed in Section 1 since the average response time can be unbounded, they cannot be expressed as non-nested limit-average automata.

Quantitative monitor automata. In verification of Boolean properties, the formalism with *monitor automata* is a very convenient way to express system properties [16]. The specification for a system can be decomposed into subproperties, and each monitor automata tracks a subproperty. Whether the system satisfies the property or not is inferred from the results of the monitor automata. Monitor automata are specially useful for safety properties, and widely used in run-time verification [13]. Our nested weighted automata framework can be seen as a natural extension of the formalism provided by monitor automata. Each slave automaton can specify a subproperty of the system, and the master automaton combines the result obtained from all the slave automata. Moreover, our nested weighted automata can be seen as the first step towards quantitative run-time verification. Each slave automaton acts as a monitor and returns values of subproperties of the system. If the value function of the master automaton is commutative (as in all our examples), the master automaton can compute an on-the-fly approximation of the value function for finite words.

5.2 Model measuring

The *model-measuring* problem [14] asks, given a model and a specification, what is the maximal distance ρ such that all models within distance ρ from the model satisfy the specification. Formally, a model M and a specification S are Boolean automata. Given M, a *similarity measure* (of M) is a function d_M from infinite words to positive real numbers such that for all traces w in $\mathcal{L}_M$ we have $d_M(w) = 0$. Similarity measures extend to models in a natural way; i.e., $d_M(M') = \sup\{d_M(w) : w \text{ is a trace of } M'\}$. The *stability radius* of S in M w.r.t. the similarity measure d_M, denoted by $\mathsf{sr}_{d_M}(M, S)$, is defined as $\mathsf{sr}_{d_M}(M, S) = \sup\{\rho \geq 0 : \forall M'(d_M(M') < \rho \Rightarrow \mathcal{L}_M \subseteq \mathcal{L}_S)\}$. We are interested in similarity measures d_M defined by nested weighted automata. Note that d_M is independent of the specification. The model-measuring decision problem of whether $\mathsf{sr}_{d_M}(M, S) \leq \lambda$ reduces to the emptiness decision question [14]. We now show how nested weighted automata can define interesting similarity measures d_M.

Example 11 (Bounded delays). Consider a model M for two parties communicating through a channel, where every sent packet is delivered in the next state. We define a similarity measure d_M, such that $d_M(w) = k$ if w obeys the rules of M, except that each packet can be delayed and the maximum delay in the trace w is k. The similarity measure d_M can be defined as follows. To obtain d_M we first construct a relaxation M_R from M in which, each sent packet is delivered after some finite time. The similarity measure is defined as an $(\text{SUP}; \text{SUM}^+)$-automaton $\mathbb{A}_D$ that computes the maximum delay. It works as follows. At each step at which a packet is sent, the master automaton starts a slave SUM^+-automaton that counts the number of steps until the packet is delivered. At steps at which no packet is sent, the master automaton takes a silent transition. The product automaton M_R and $\mathbb{A}_D$ defines the desired similarity measure (see Appendix for further details).

5.3 Model repair

The *model-repair* problem, given a model and a specification, asks for the minimal restriction of the model such that the specification is satisfied. Given a model M, a *repair measure* d_M is a function from infinite words to real numbers such that $d_M(w) < \infty$ iff $w \in \mathcal{L}_M$. Intuitively, the measure evaluates the hardness of traces of M, which can be used to evaluate severity of the violation of the specification. We are interested in d_M specified by nested weighted automata. Given a model M, a repair measure d_M, and a real number r, we define the language $d_M^{<r}$ as $\{w : d_M(w) < r\}$. The model-repair decision problem, given a model M, a repair measure d_M, and a specification S, asks whether $\sup\{r : d_M^{<r} \subseteq \mathcal{L}_S\} \leq \lambda$. The model-repair decision problem also reduces to the emptiness question.

Example 12 (Context-switches). Consider a system consisting of a scheduler and two programs. The scheduler starts processes infinitely often and does preemptive scheduling. To obtain a finite-state model, we consider that only a single instance of each program may run at a given time. Consider the repair measure d_M that represents the negative of the *minimal slot length*, i.e., for all w we have $d_M(w) = -k$ iff each process in the execution w, runs for at least k steps. The repair measure can be defined by a functional $(\text{SUP}; \text{SUM})$-automaton $\mathbb{A}_R$ as follows. After each context-switch, the master automaton starts an automaton that computes the running time until the next context-switch and multiplies it by -1 (i.e., add -1 at each step). At steps at which there is no context switch, the master automaton takes a silent transition. It follows that the supremum of all those values is the length of the shortest running time of a process multiplied by -1. Although, the emptiness

problem is undecidable for (SUP; SUM)-automata, the automaton $\mathbb{A}_R$ has only non-positive weights. The emptiness problem for (SUP; SUM)-automata with non-positive weights reduces to the universality problem for (INF; SUM$^+$)-automata, which is decidable.

6 Conclusion and Future Work

In this work we introduced the framework of nested weighted automata as a new and expressive formalism to specify quantitative properties. We studied the decidability of the basic decision questions. There are several directions for future work. First, we have an open conjecture (Conjecture 9) regarding the decidability of the emptiness of non-deterministic (LIMAVG; SUM)-automata. Second, another interesting direction would be to establish optimal complexity results for the decision problems. Third, there are several possible extensions of the nested weighted automata model, such as, considering (i) two-way master and slave automata, (ii) multiple levels of nesting; and (iii) instead of infimum across paths consider average measures across paths (i.e., probability distribution over runs and expected value of the runs as in [6]).

References

1. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *ATVA*, pages 482–491. LNCS 6996, Springer, 2011.
2. R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.
3. B. Bollig, P. Gastin, B. Monmege, and M. Zeitoun. Pebble weighted automata and transitive closure logics. In *ICALP (2)*, pages 587–598. LNCS 6199, Springer, 2010.
4. K. Chatterjee, L. Doyen, H. Edelsbrunner, T. A. Henzinger, and P. Rannou. Mean-payoff automaton expressions. *CoRR*, abs/1006.1492, 2010.
5. K. Chatterjee, L. Doyen, and T. A. Henzinger. Alternating weighted automata. In *FCT'09*, pages 3–13. Springer-Verlag, 2009.
6. K. Chatterjee, L. Doyen, and T. A. Henzinger. Probabilistic weighted automata. In *CONCUR*, pages 244–258. LNCS 5710, Springer, 2009.
7. K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. *Logical Methods in Computer Science*, 6(3), 2010.
8. K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.
9. A. Degorre, L. Doyen, R. Gentilini, J.-F. Raskin, and S. Torunczyk. Energy and mean-payoff games with imperfect information. In *CSL*, pages 260–274. LNCS 6247, Springer, 2010.
10. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.
11. J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, USA, 1996.
12. E. Filiot, R. Gentilini, and J.-F. Raskin. Quantitative languages defined by functional automata. In *CONCUR*, pages 132–146. LNCS 7454, Springer, 2012.
13. K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*, pages 342–356. LNCS 2280, Springer, 2002.
14. T. A. Henzinger and J. Otop. From model checking to model measuring. In *CONCUR*, pages 273–287. LNCS 8052, Springer, 2013.
15. M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
16. A. Pnueli and A. Zaks. On the merits of temporal testers. In *25 Years of Model Checking*, pages 172–195. LNCS 5000, Springer, 2008.

A Proofs from Section 4

Theorem 2. *(1) For $f \in \mathsf{InfVal}$, the universality problem for $(f; \text{SUM})$-automata is undecidable. (2) For $g \in \mathsf{FinVal}$, the universality problem $(\text{LIMAVG}; g)$-automata is undecidable.*

Proof (of (1)). We show a reduction of the universality problem for SUM-automata with weights $\{-1, 0, 1\}$, which is undecidable (Theorem 1), to the universality problem for $(f; \text{SUM})$-automata, where $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$. The case $f = \text{LIMAVG}$ follows from (2).

Let $\mathcal{A}$ be a SUM-automaton with weights $\{-1, 0, 1\}$. Consider an $(\text{INF}; \text{SUM})$-automaton $\mathbb{A}$ that works as follows. Its acceptance condition enforces that it accepts only words with infinitely many $\#$ letters, i.e., the words the form $w_1 \# w_2 \# \ldots$. At each $\#$ letter the master automaton starts an instance of $\mathcal{A}$ as a slave automaton that works to the successive $\#$ letter. On the positions with a letter different than $\#$, the master automaton takes a silent transition. Then, the value of a word $w_1 \# w_2 \# \ldots$ is equal to the infimum of values $\mathcal{L}_{\mathcal{A}}(w_i)$. In particular, $\mathcal{L}_{\mathbb{A}}((w\#)^\omega) = \mathcal{L}_{\mathcal{A}}(w)$. It follows that the universality problems for $\mathbb{A}$ and $\mathcal{A}$ coincide.

The same construction shows a reduction of the universality problem for SUM-automata to the universality problem for $(\text{LIMINF}; \text{SUM})$-automata (resp. $(\text{SUP}; \text{SUM})$-automata, $(\text{LIMSUP}; \text{SUM})$-automata). □

Proof (of (2)). For every $g \in \mathsf{FinVal}$ we can define two dummy g-automata, $\mathcal{A}_0$ (resp. $\mathcal{A}_1$) that immediately accept and return the value 0 (resp. 1). Therefore, such $(\text{LIMAVG}; g)$-automata can simulate all LIMAVG-automata with weights $0, 1$, whose universality problem is undecidable (Theorem 1). Therefore, the universality problem for $(\text{LIMAVG}; g)$-automata is undecidable as well. □

B Proofs from Section 4.1

In the following key lemma we establish that if the slave automata are regular weighted automata, then nested weighted automata can be reduced to weighted automata with the same value function as for the master automata. For regular weighted slave automata, a weighted automaton can simulate a nested automaton in the following way. Instead of starting a slave automaton, the weighted automaton guesses the weight of the current transition (i.e., the value to be returned of the slave automaton) and checks that the guessed weight is correct. The definition of regular weighted automata implies that such a check can be done by a (non-weighted) finite automaton $\mathcal{S}$. Thus, the weighted automaton takes a universal transition such that in one branch it continues its execution and in another it runs $\mathcal{S}$. Observe that such a universal transition can be removed by a standard power-set construction.

Lemma 4 (Key reduction lemma). *Let $f \in \mathsf{InfVal}$ be a value function. Consider a nested weighted automaton $\mathbb{A} = \langle \mathcal{A}_{mas}; f; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$ such that all automata $\mathfrak{B}_1, \ldots, \mathfrak{B}_k$ are regular weighted automata. There is an $\mathsf{sil}(f)$-automaton $\mathcal{A}$ (weighted automaton), that can be constructed in exponential time, which is equivalent to $\mathbb{A}$; moreover, if $\mathbb{A}$ is functional, then $\mathcal{A}$ is functional as well.*

Proof. Assume that each slave automaton $\mathfrak{B}_i$ has the weights from the set $\{-n, \ldots, n\}$. Then, since all of the slave automata are regular weighted automata, for all $i \in \{1, \ldots, k\}$ and $j \in \{-n, \ldots, n\}$ there is a deterministic finite word automaton $\mathcal{S}_{i,j}$ that recognizes the

language of all words w such that $\mathcal{L}_{\mathfrak{B}_i}(w) = j$. Since $\mathfrak{B}_i$ is a regular weighted automaton, it accepts precisely when one of the automata $\mathcal{S}_{i,0}, \ldots, \mathcal{S}_{i,n}$ accepts.

We define Q_S (resp. F_S) as the disjoint union of the sets of states (resp. the sets of accepting states states) of all automata $\mathcal{S}_{i,j}$. Let Q_m (resp. F_m) be the set of all states (resp. all accepting states) the master automaton $\mathcal{A}_{mas}$. We define a relation $\textsc{Step} \subseteq 2^{Q_S} \times \Sigma \times 2^{Q_S}$, which is the union of transition relations lifted to sets of states, i.e., $(\{q_1, \ldots, q_l\}, a, \{q'_1, \ldots, q'_l\}) \in \textsc{Step}$ iff for every $m \in \{1, \ldots, l\}$, some automaton $\mathcal{S}_{i,j}$ has a transition (q_m, a, q'_m).

We define $\mathcal{A}$, which we show is equivalent to $\mathbb{A}$, as a generalized Büchi automaton, which differs from an automaton over infinite words (Büchi automaton) in the acceptance condition. An acceptance condition in a generalized Büchi automaton is a sequence of $F_1, \ldots, F_s$ of sets of states. A run is accepting iff for each $d \in \{1, \ldots, s\}$ there is a state from F_d visited infinitely often. There is a straightforward reduction of a generalized Büchi automaton to a Büchi automaton, and we omit the reduction and for technical convenience consider generalized Büchi condition for the proof.

The automaton $\mathcal{A}$ works as follows. It simulates the execution of the master automaton. Every time the master automaton starts a slave automaton $\mathfrak{B}_i$, the automaton guesses the value j that $\mathfrak{B}_i$ returns and checks it, i.e., it starts simulating the automaton $\mathcal{S}_{i,j}$, by including the initial state of $\mathcal{S}_{i,j}$ in a set of states P_1. The automaton $\mathcal{A}$ maintains two sets of states of simulated automata, P_1 and P_2: states in P_1 and P_2 represent states of $\mathcal{S}_{i,j}$ and basically, there are states in P_2 until all automata corresponding to them terminate. Once they do, P_2 is empty and all states from P_1 are copied to P_2. Intuitively, the role of P_1 and P_2 is to ensure that each automaton terminates, by enforcing P_2 to be empty infinitely often. We now formally define $\mathcal{A} = \langle \Sigma, Q, q_0, C, \delta, F \rangle$ as follows:

1. $Q = Q_m \times (\{-n, \ldots, n\} \cup \{\bot\}) \times 2^{Q_S} \times 2^{Q_S}$
2. $q_0 = \langle q_0^m, 0, \emptyset, \emptyset \rangle$, where q_0^m is the initial state of the master automaton
3. $(\langle q, j, P_1, P_2 \rangle, a, \langle q', j', P'_1, P'_2 \rangle) \in \delta$ iff (q, a, q') is a valid transition of the master automaton labeled by i and one of the following holds (intuitive descriptions follow):
   (a) $j = \bot$, $P'_1 = P''_1 \setminus F_S, P'_2 = P''_2 \setminus F_S$, where $\textsc{Step}(P_1, a, P''_1)$ and $\textsc{Step}(P_2, a, P''_2)$,
   (b) $j \neq \bot$, $P_2 = \emptyset$, $P'_1 = \{q_0^{i,j}\}$ and $P'_2 = P''_2 \setminus F_S$, where $\textsc{Step}(P_1, a, P''_2)$ and q_0^i is the initial state of $\mathcal{S}_{i,j}$, the automaton that checks that the slave automaton $\mathfrak{B}_i$ started at the current position returns the value j,
   (c) $j \neq \bot$, $P_2 \neq \emptyset$, $P'_1 = (P''_1 \cup \{q_0^{i,j'}\}) \setminus F_S$ and $P'_2 = P''_2 \setminus F_S$, where $\textsc{Step}(P_1, a, P''_1)$ and $\textsc{Step}(P_2, a, P''_2)$

   The intuitive descriptions are as follows: (a) the first transition corresponds to a silent transition, and hence we compute the successor states of sets P_1 and P_2 and remove the accepting states (that correspond to automata that terminate); (b) the second transition is similar to the first case but here a new automaton that simulates the slave automaton is started, but since P_2 is empty we compute the next P'_2 from the successor of P_1 according to $\textsc{Step}$ but after removing the accepting states, and the new P_1 is the initial state of the simulating automaton; and (c) the third transition is very similar to the first transition just that the initial state of the simulating automaton is added to the P'_1.
4. the cost function is defined as $C(\langle q, j, P_1, P_2 \rangle, a, \langle q', j', P'_1, P'_2 \rangle) = j'$,
5. F consists of $F_1 = F_m \times (\{-n, \ldots, n\} \cup \{\bot\}) \times 2^{Q_S} \times 2^{Q_S}$ and $F_2 = Q_m \times (\{-n, \ldots, n\} \cup \{\bot\}) \times 2^{Q_S} \times \emptyset$. Intuitively, F_1 ensures that the acceptance condition of the master automaton is satisfied and F_2 ensures that P_2 is empty infinitely often.

In the remaining part we shall prove correctness of the construction by showing that for every infinite word w

(i) for every run of $\mathbb{A}$ on w of the value x, there is a run of $\mathcal{A}$ of the value x, and
(ii) for every run of $\mathcal{A}$ of the value x, there is a run of $\mathbb{A}$ of the value x.

(i): Assume that $(\Pi, \pi_1, \pi_2, \ldots)$ is a run of $\mathbb{A}$ on w of the value x. We construct inductively a sequence r that corresponds $(\Pi, \pi_1, \pi_2, \ldots)$, which will be transformed into a run γ of $\mathcal{A}$. For each π_s, which is a run of some slave automaton $\mathfrak{B}_i$ of the value j, we define $p[s]$ as the position in w at which the run π_s starts and η_s as a minimal length accepting run of the automaton $\mathcal{S}_{i,j}$ on some subword of $w[p[s], s']$. The run η_s contains only a single accepting state, which is the last state.

Now, we define a pre-run r, which is a sequence of quadruples of a state of the master automaton, the current weight and two sets of runs of $\mathcal{S}_{i,j}$. We define r by induction. First, $r[0]$ is defined as $(q_0^m, 0, \emptyset, \emptyset)$, where q_0^m is the initial state of $\mathcal{A}_{mas}$. For $l \geq 0$, given $r[l] = \langle q, j, \mathcal{G}, \mathcal{H} \rangle$, we define $r[l+1]$ as follows: (intuitively, $\mathcal{G}$ (resp. $\mathcal{H}$) represent sets of runs which will later be converted to sets of states for P_1 (resp. P_2))

1. If the master automaton takes a silent transition, then $r[l+1] = \langle \Pi[l+1], \bot, \mathcal{G}', \mathcal{H}' \rangle$, such that $\mathcal{G}' = \mathcal{G}'' \setminus F_S$ and $\mathcal{H}' = \mathcal{H}'' \setminus F_S$, where $\mathcal{G}'$ (resp. $\mathcal{H}'$) consists of the runs from $\mathcal{G}$ (resp. $\mathcal{H}$) where each run has the first element removed.
2. If the master automaton takes a transition labeled with i, then $r[l+1] = \langle \Pi[l+1], j', \mathcal{G}', \mathcal{H}' \rangle$, where s is such that $l = p[s]$, j' is the value of π_s and $\mathcal{G}', \mathcal{H}'$ are defined as follows:
   (a) If $\mathcal{H} = \emptyset$, $\mathcal{G}' = \{\eta_s\}$ and $\mathcal{H}' = \mathcal{H}'' \setminus F_S$, where $\mathcal{H}''$ consists of the runs from $\mathcal{G}$ with the first element removed.
   (b) If $\mathcal{H} \neq \emptyset$, $\mathcal{G}' = \mathcal{G}'' \setminus F_S$ and $\mathcal{H}' = \mathcal{H}'' \setminus F_S$, where $\mathcal{G}''$ consists of the runs from $\mathcal{G}$ with the first element removed and $\mathcal{H}''$ consists of the runs from $\mathcal{H}$ with the first element removed and the run η_s.

Now, we define γ from r by projecting runs in the third and fourth components on the first element, i.e., for every $l \geq 0$, where $r[l] = \langle q, j, \mathcal{G}, \mathcal{H} \rangle$, we define $\gamma[l]$ as $\langle q, j, P_1, P_2 \rangle$, where P_1 (resp. P_2) are the sets of first elements of runs in $\mathcal{G}$ (resp. $\mathcal{H}$). One can easily check that γ is an accepting run of $\mathcal{A}$. Moreover, each transition of γ has the same weight as the corresponding transition of the master automaton in $\mathbb{A}$. Hence, the value of γ is x, the same as the value of $(\Pi, \pi_1, \pi_2, \ldots)$.

(ii): Assume that $\mathcal{A}$ has a run β of the value x. First, observe that Π defined as a projection of β on the first component is a valid accepting run of $\mathcal{A}_{mas}$. We shall prove that for every $l \geq 0$ such that l-th transition $\langle \Pi[l], w[l], \Pi[l+1] \rangle$ is labeled by $i \in \{0, \ldots, k\}$, i.e., it is non-silent, there is a run of $\mathfrak{B}_i$ of the weight j, where $\beta[l+1] = \langle \Pi[l+1], j, P_1, P_2 \rangle$. Then, we can readily conclude that $\mathbb{A}$ has a run of the value x. Consider such a position $l \geq 0$. Since the transition taken by $\mathcal{A}$ has the weight j, the state $q_0^{i,j}$ belongs P_1 in $\beta[l+1] = \langle q, j, P_1, P_2 \rangle$. One can track the descendant states of $q_0^{i,j}$ and notice that some descendant of $q_0^{i,j}$ belongs to F_S. Indeed, if $q_0^{i,j}$ has no descendant from F_S, it would have infinitely many descendants, which would prevent the fourth component P_2 be empty infinitely often.

Functionality: Assume that $\mathbb{A}$ is functional. Consider a word w and two runs of $\mathcal{A}$ a word w. Denote by c_1, c_2 the values of these runs. Then, $\mathbb{A}$ has also runs of values c_1, c_2 on w. Due to functionality of $\mathbb{A}$ we have $c_1 = c_2$. It follows that $\mathcal{A}$ is functional. □

Now, we prove two very simple lemmata regarding weighted automata with silent moves.

Lemma 13. *Let $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$. For every $\mathsf{sil}(f)$-automaton $\mathcal{A}$ there is an (computable in linear time) f-automaton $\mathcal{A}'$ equivalent to $\mathcal{A}$.*

Proof. Given an $\mathsf{sil}(f)$-automaton $\mathcal{A}$, where $f \in \mathsf{InfVal}$, we define the automaton $\mathcal{A}^{\ell}$ as the f-automaton that results from $\mathcal{A}$ by substituting each silent transition by a transition of the weight ℓ. Observe that for every $\mathsf{sil}(\text{INF})$-automaton $\mathcal{A}$ for every infinite word w we have $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$ iff $\mathcal{L}_{\mathcal{A}^{(\lambda+1)}}(w) \leq \lambda$. The same equivalence holds for every $\mathsf{sil}(\text{SUP})$-automaton $\mathcal{A}$ and its variant $\mathcal{A}^{(\lambda-1)}$. Thus, the emptiness and universality problems for $\mathsf{sil}(\text{INF})$-automata (resp. $\mathsf{sil}(\text{SUP})$-automata) and INF-automata (resp. SUP-automata) coincide. Now, a run of an $\mathsf{sil}(\text{INF})$-automaton is accepting only if it contains infinitely many non-silent transitions. Therefore, the above equivalences hold for $f \in \{\text{LIMINF}, \text{LIMSUP}\}$ and the corresponding problems coincide. □

Lemma 14. *The emptiness problem for* $\mathsf{sil}(\text{LIMAVG})$*-automata is decidable.*

Proof. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F, C)$ be a $\mathsf{sil}(\text{LIMAVG})$-automaton. We define a LIMAVG-automaton $\mathcal{A}_{\text{fix}}$ such that for every accepting run η of $\mathcal{A}$, the run η', resulting from η by removing silent transitions, is an accepting run of $\mathcal{A}_{\text{fix}}$, and vice versa, every accepting run of $\mathcal{A}_{\text{fix}}$ can be extended to a run of $\mathcal{A}$ by inserting silent transitions. The set of states of $\mathcal{A}_{\text{fix}}$ is the same as $\mathcal{A}$ and the transition relation of $\mathcal{A}_{\text{fix}}$ consists of (q_1, a, q_2) such that there is $(q_1', a, q_2') \in \delta$ and q_1' (resp. q_2) is reachable from q_1 (resp. q_2') by a path consisting of only silent transitions. The weight of such a transition is the infimum over the weights of transitions $(q_1', a, q_2') \in \delta$ that generate (q_1, a, q_2). If follows from the construction that $\mathcal{A}_{\text{fix}}$ has the stipulated properties and their optimal runs have the same value. Therefore, the emptiness problem for $\mathsf{sil}(\text{LIMAVG})$-automata reduces in polynomial time to the emptiness problem for LIMAVG-automata. □

Theorem 5. *Let* $g \in \{\text{MIN}, \text{MAX}, \text{SUM}^B\}$*. The following assertions hold: (1) The emptiness and universality problems are decidable for non-deterministic* $(f; g)$*-automata, where* $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$*. (2) The emptiness problem is decidable for non-deterministic* $(\text{LIMAVG}; g)$*-automata. (3) The universality problem is decidable for functional* $(\text{LIMAVG}; g)$*-automata.*

Proof. (1): Let $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$ and $g \in \{\text{MIN}, \text{MAX}, \text{SUM}^B\}$. Due to Lemma 4 and Lemma 13, every $(f; g)$-automaton is equivalent to some f-automaton. The emptiness and universality problems are decidable for f-automata (Theorem 1).

(2): Lemma 4 state that $(\text{LIMAVG}; g)$-automata are equivalent to $\mathsf{sil}(\text{LIMAVG})$-automata, which enjoy decidability of the emptiness problem (Lemma 14).

(3): The universality problem for functional $(\text{LIMAVG}; g)$-automata reduces to the emptiness problem for functional $(\text{LIMAVG}; g)$-automata. It suffices to take inverses of all weights in all slave automata of a given nested automaton. □

C Proofs from Section 4.2

Theorem 6 (Crucial undecidability result). *(1) The emptiness problem for deterministic* $(\text{SUP}; \text{SUM})$*- and* $(\text{LIMSUP}; \text{SUM})$*-automata is undecidable. (2) The universality problem for deterministic* $(\text{INF}; \text{SUM})$*- and* $(\text{LIMINF}; \text{SUM})$*-automata is undecidable.*

Proof (of (1)). Given a Minsky's machine $\mathcal{M}$, we construct a deterministic $(\text{SUP}; \text{SUM})$-automaton $\mathbb{A}$ that accepts infinite words of the form $w_1 \# w_2 \# \ldots$. Moreover, the value of the word $w_1 \# w_2 \# \ldots$ is 0 iff each subword w_i encodes an valid accepting computation of $\mathcal{M}$. As the problem, given a Minsky's machine, does it have an accepting computation is undecidable, we conclude that the emptiness problem for deterministic $(\text{SUP}; \text{SUM})$-automata (resp. $(\text{LIMSUP}; \text{SUM})$-automata) is undecidable.

A Minsky's machine $\mathcal{M}$ is a finite automaton augmented with two counters c_1, c_2. The counters can be incremented, decremented and tested whether they are zero or positive. The transitions of $\mathcal{M}$ depend on the values of counters, namely, whether they are equal zero. That is, each transition has the following form $(q, s, t) \to (q', v_1, v_2)$, where $s \in \{\mathrm{c}_1 = 0, \mathrm{c}_1 > 0\}, t \in \{\mathrm{c}_2 = 0, \mathrm{c}_2 > 0\}$ and $v_1, v_2 \in \{-1, 0, 1\}$. E.g. $(q, \mathrm{c}_1 = 0, \mathrm{c}_1 > 0) \to (q', +1, -1)$ means that if the machine is in the state q, the value of c_1 is 0 and c_2 greater than 0, then the next state is q', c_1 is incremented and c_2 is decremented.

We define two notions for Minsky's machines, a *run* and a *computation*. A *run* of a Minsky's machine $\mathcal{M}$ is a sequence $(q_0, 0, 0), (q_1, \alpha_1, \beta_1), \ldots, (q_n, \alpha_n, \beta_n)$ such that for every $i < n$ there is a transition of $\mathcal{M}$ $(q_i, s, t) \to (q_{i+1}, v_1, v_2)$ such that α_i satisfies s β_i satisfies t, and $\alpha_{i+1} = \alpha_i + v_1$, $\beta_{i+1} = \beta_i + v_2$. A run is *accepting* iff its last element is $(q_F, 0, 0)$. A *computation* of $\mathcal{M}$ is a sequence of elements $Q \times \{\mathrm{c}_1 = 0, \mathrm{c}_1 > 0\} \times \{\mathrm{c}_2 = 0, \mathrm{c}_2 > 0\} \times \{-1, 0, 1\} \times \{-1, 0, 1\}$ called *configurations*. A computation $(q_0, \mathrm{c}_1 = 0, \mathrm{c}_2 = 0, 0, 0), (q_1, t_1, \beta_1, x_1, y_1), \ldots, (q_n, \mathrm{c}_1 = 0, \mathrm{c}_2 = 0, x_n, y_n)$ is *valid* iff there is an accepting run $(q_0, 0, 0), \ldots, (q_n, \alpha_n, \beta_n)$ such that for every $i \in \{0, \ldots, n\}$, $\alpha_i = \sum_{j=0}^{i} x_j$ and $\beta_i = \sum_{j=0}^{i} y_j$.

Consider a valid computation η and the corresponding accepting run π. For positions $i < j$, let *balance* of c_1 (resp. c_2) between position i and j (in η) denote the difference in the number of increments and decrements of c_1 (resp. c_2) between i and j. Since the initial value of the counters is 0, the value of a counter c_1 (resp. c_2) in $\pi[i]$ is precisely its balance of c_1 (resp. c_2) between positions 1 and i. Thus, a zero test (non-zero test) at the position i is valid iff the balance between positions 1 and i is 0 (is strictly positive).

Consider a computation η of a Minsky machine $\mathcal{M}$. If it is invalid then there is a first position in η such that the corresponding sequence over $Q \times \mathbb{N} \times \mathbb{N}$ is not a run. There a basically, two reasons for that: (i) $\mathcal{M}$ has no transition consistent with a step from $\eta[i]$ to $\eta[i+1]$, (ii) the configuration at $\eta[i]$ is inconsistent with the current values of c_1, c_2, i.e., a zero or a non-zero test is inconsistent with the actual value of a counter. A Boolean automaton can check whether the computation is invalid because of (i). We show how to check (ii), i.e., validity of zero and non-zero tests, using a nested weighted automaton. We will discuss only the first counter c_1 as the construction for c_2 is virtually the same.

First, we check validity of zero tests on c_1. All zero tests on c_1 are valid iff the balance of c_1 between any two consecutive zero tests is zero. To check that this holds, the nested automaton starts at each position i with a zero test two deterministic slave SUM-automata: $\mathcal{A}^{+}_{\mathrm{c}_1=0}, \mathcal{A}^{-}_{\mathrm{c}_1=0}$. The automaton $\mathcal{A}^{+}_{\mathrm{c}_1=0}$ computes the balance of c_1 between i and the next zero test of c_1; it increments (decrements) its value whenever c_1 is incremented (decremented), and it terminates at the next zero test of c_1. The automaton $\mathcal{A}^{-}_{\mathrm{c}_1=0}$ does the opposite, i.e., it computes the additive inverse of the balance between i and the next zero test of c_1. The values of these automata are inverses of each other and the maximum of their values is the absolute value of the balance. Hence, the maximum of their values is less-or-equal to zero iff the balance between i and the next zero test of c_1 is 0. Thus, the values of all slave automata $\mathcal{A}^{+}_{\mathrm{c}_1=0}, \mathcal{A}^{-}_{\mathrm{c}_1=0}$ are less-or-equal to zero if and only if all zero tests of c_1 are valid.

Second, we check that non-zero tests are valid. To do that, the automaton starts at every position i with a non-zero test a third slave SUM-automaton $\mathcal{A}_{\mathrm{c}_1>0}$ that first increments its value to 1 and then computes the balance between i and the next zero test. The value of c_1 at the position i is strictly greater than 0 iff the balance between the position i and the next position at which that counter is 0 does not exceed -1. Provided that verifying zero tests

succeeds, the value of $\mathcal{A}_{c_1>0}$ is less-or-equal to 0 iff the non-zero test at the position i is valid.

The value of the nested automaton does not exceed 0 if and only if the values of all slave automata are less-or-equal to 0, which holds precisely when all zero and non-zero tests on c_1 are valid. A similar construction can be repeated for the counter c_2. In the above construction up to four automata has to be started at any configuration, while nested automata can start at most one slave automaton at each step. However, we can encode configurations by some fixed number of letters. E.g. $c\ \$\ \$\ \$\ \$$ where c is a letter that fully encodes a configuration (q, α, β, x, y) and $\$$ letters are used only to start enough slave automata. It follows that $\mathbb{A}$ accepts a word $w_1\#w_2\#\ldots$ and assigns it the value 0 iff each word w_i encodes an valid accepting computation of $\mathcal{M}$.

Observe that the same automaton, $\mathbb{A}$, considered as (LIMSUP; SUM)-automaton returns the same result. Indeed, if a given Minsky's machine does not have an accepting computation, each accepted word will have positive value. On the other hand, if there is an accepting computation w, the value of (SUP; SUM)-automata and (LIMSUP; SUM)-automata the word $(w\#)^\omega$ coincides, hence it is 0. □

Proof (of (2)). The universality problem for deterministic (INF; SUM)-automata is the dual of the emptiness problem for deterministic (SUP; SUM) automata. Indeed, consider a deterministic (INF; SUM)-automaton $\mathbb{A}$ and the nested automaton $\mathbb{A}'$ that results from taking inverses of all weights in $\mathbb{A}$ and changing its value function to INF. One can easily check that for every word w, the weight of w assigned by $\mathbb{A}$ is x, then $\mathbb{A}'$ assigns to w the weight $-x$. □

D Proofs from Section 4.3

We prove Theorem 7 and the proofs are relatively straightforward.

Theorem 7. *(1) The emptiness problem for* (INF; SUM)*-automata and* (LIMINF; SUM)*-automata is decidable. (2) The universality problem for functional* (SUP; SUM)*-automata and* (LIMSUP; SUM)*-automata is decidable. (3) For* $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$*, the emptiness and the universality problems for* $(f; \text{SUM}^+)$*-automata are decidable.*

Proof (of (1)). Let $\mathbb{A} = \langle \mathcal{A}_{mas}; \text{INF}; \mathfrak{B}_1, \ldots, \mathfrak{B}_k \rangle$ be an (INF; SUM)-automaton. We construct a SUM-automaton over finite words $\mathcal{A}$ such that the emptiness problem for $\mathbb{A}$ and $\mathcal{A}$ coincide. The automaton $\mathcal{A}$ works over words over the alphabet $\Sigma \cup \{\#, 1, \ldots k\}$ of the form $wiv\#u'\#u$, where $w, v, u, u' \in \Sigma^*$ and $i \in \{1, \ldots k\}$, and the value of its run, if it is accepting, is the value of the slave automaton $\mathfrak{B}_i$ on the word v. The automaton $\mathcal{A}$ consists of two components. The first, a Boolean one whose all weights are 0, ensures that $\mathbb{A}$ has an accepting run on $wvu'u^\omega$ such that the slave automaton started at the beginning of the word v is $\mathfrak{B}_i$ and $\mathfrak{B}_i$ accepts the word v. The second component is a weighted one and it computes the value of $\mathfrak{B}_i$ on v. Observe that the value of each run of $\mathbb{A}$ depends only on a finite prefix of a word, i.e., for each run of $\mathbb{A}$ there is a finite prefix $wvu'u$ such that the value of that run equals $\mathcal{L}_{\mathcal{A}}(wiv\#u'\#u)$. It follows that the emptiness problem for $\mathbb{A}$ and $\mathcal{A}$ coincide. □

Proof (of (2)). The emptiness problem for functional INF-SUM automata is the dual of the universality problem for functional SUP-SUM automata. It suffices to take inverses of all weights. Thus, (2) follows from (1). □

Proof (of (3) and (4)). Let λ be the threshold given in the emptiness (resp. universality) problem. Observe that for $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$, for every word w, an $(f; \text{SUM}^+)$-automaton has a run of the value not exceeding λ threshold iff $(f; \text{SUM}^B)$-automaton, where $B = \lambda + 1$, has a run of the value not exceeding λ. It follows that the emptiness (resp. universality) problem for $(f; \text{SUM}^+)$-automata reduces to the emptiness (resp. universality) problem for $(f; \text{SUM}^B)$-automata, where B varies.

Since SUM^B is a regular value function, and regular languages certifying it can be effectively in linear time in B, Lemma 4 implies that for $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$, the universality problem for $(f; \text{SUM}^+)$-automata reduces (in exponential time) to the universality problem for $\mathsf{sil}(f)$-automata, which is decidable (Lemma 13). □

E The proof of Theorem 8

Theorem 8. *The emptiness problem for* $(\text{LIMAVG}; \text{SUM}^+)$*-automata is decidable; and the universality problem for functional* $(\text{LIMAVG}; \text{SUM}^+)$*-automata is decidable.*

We prove decidability of the emptiness and the universality problem in separate subsections.

E.1 The proof of decidability of the emptiness problem for $(\text{LIMAVG}; \text{SUM}^+)$-automata.

Lemma 15. *The emptiness problem for* $(\text{LIMAVG}; \text{SUM}^+)$*-automata reduces to the emptiness problem for deterministic* $(\text{LIMAVG}; \text{SUM}^+)$*-automata.*

Proof. The emptiness problem can be regarded as a single player game who chooses letters and transitions to satisfy the objective. Thus, we can extend the alphabet such that each letter uniquely determines transitions for the master and slave automata. This is the key intuition to reduce the emptiness question to one for deterministic automata. The determinization by extension of alphabet enforces that the following condition on runs holds: (Cond *) all slave automata that at a position s in a word are in the same state, take the same transition. We need to show that imposing (Cond *) does not affect the value of optimal runs.

Consider $(\text{LIMAVG}; \text{SUM}^+)$-automaton $\mathbb{A}$ and its optimal run $(\Pi, \pi_1, \pi_2, \dots)$. If $\mathbb{A}$ does not have an accepting run of finite value, we take any accepting run. Consider π_i, π_j that are in the same state at the position s in the word, i.e., $\pi_i[i'] = \pi_j[j']$, where i', j' are the position is π_i, π_j corresponding to the position s in w. We choose from the suffixes $\pi_i[i', |\pi_i|], \pi_j[j', |\pi_j|]$ the one with the smaller value and change the suffixes of both runs to the chosen one. If these suffixes have the same value, we chose the shorter one. Such a transformation does not increase the value of the partial sums and does not introduce infinite runs of slave automata. Indeed, a run of each slave automaton can be changed by such an operation only finitely many times. Thus, this transformation can be applied to any pair of slave runs to obtain an optimal run satisfying (Cond *). □

Given a deterministic nested automaton $\mathbb{A}$, we define Q_s as the disjoint union of the sets of states of all slave automata of $\mathbb{A}$. For an infinite word w, we say that (q_m, A) is the *configuration* at the position p in w if q_m is the state of the master automaton of $\mathbb{A}$ at the position p and $A \subseteq Q_s$ is the set of states of all slave automata at the position p. We denote by $\mathsf{conf}(\mathbb{A})$ the number of configurations of $\mathbb{A}$. We define the *multiplicity* mult

at the position p as the function $\mathsf{mult} : A \mapsto \mathbb{N}$, where A is as above, such that $\mathsf{mult}(q)$ specifies the number of slave automata in the state q at the position p. The configuration together with the multiplicity give a complete description of the state of $\mathbb{A}$ at the position p. We define $\mathcal{C}_{w[1,p]}, \mathsf{mult}_{w[1,p]}$ as the configuration and the multiplicity at the position p. Since $\mathbb{A}$ is deterministic, $\mathcal{C}_{w[1,p]}, \mathsf{mult}_{w[1,p]}$ are uniquely determined (and the same) for all words $w[1,p]w'$.

Definition 16. *A run of an* (LIMAVG; SUM$^+$)*-automaton is of* width bounded by c *iff at each position the number of running slave automata is bounded by* c*.*

Lemma 17. *Let* $\mathbb{A}$ *be a deterministic* (LIMAVG; SUM$^+$)*-automaton that recognizes a non-empty language. There is a constant* $\mathbf{c}$ *exponentially bounded in* $|\mathbb{A}|$ *and a deterministic* (LIMAVG; SUM$^+$)*-automaton* $\mathbb{A}_0$ *equivalent to* $\mathbb{A}$ *that has an optimal run of width bounded by* $\mathbf{c}$*. The maximal weight of* $\mathbb{A}_0$ *is bounded by* $2 \cdot \mathbf{c}$*.*

As in deterministic each word has the unique run, so we call an infinite word w *optimal* iff the run on w is optimal. Before we prove Lemma 17, we show its vital component:

Lemma 18. *Let* $\mathbb{A}$ *be a deterministic* (LIMAVG; SUM$^+$)*-automaton that recognizes a non-empty language. Let* $\mathbf{N} = (|Q_s| + 2) \cdot \mathsf{conf}(\mathbb{A})$*. There is an optimal word of* $\mathbb{A}$ *such that for some* $s_0 > 0$*, at each position* $s > s_0$ *at most* $2 \cdot \mathbf{N}$ *slave automata will accumulate (past the position* s*) value grater than* $4 \cdot \mathbf{N}$*.*

Proof. For a multiplicity mult we define its restriction to $\mathbf{N}$, $\mathsf{mult} \restriction_{\mathbf{N}}$, as $\mathsf{mult} \restriction_{\mathbf{N}} (q) = \min(\mathsf{mult}(q), \mathbf{N})$, for every $q \in \mathrm{dom}(\mathsf{mult})$.

Consider any optimal word uw such that at the position $|u|$ there are $2 \cdot \mathbf{N}$ slave automata that will accumulate in w (past the position $|u|$ in uw) value grater than $4 \cdot \mathbf{N}$. We show a transformation of uw to uw', such that uw' has the same value and at the position $|u|$ no slave automaton will accumulate in w' value grater than $4 \cdot \mathbf{N}$. Let $j_0 > |u|$ be a position in uw with an accepting state and let $j_1, \ldots, j_n$ be the positions at which each of slave automata started before the position $|u|$ finishes. Note that $n \leq |Q_s|$. As slave automata work on finite words such $j_1, \ldots, j_n$ exist. Finally, let j be the first position greater than $max(j_0, j_1, \ldots, j_n)$ with the configuration $\mathcal{C}_{uw[1,j-|u|]} = \mathcal{C}_u$ and multiplicity $\mathsf{mult}_{uw[1,-|u|]} \restriction_{\mathbf{N}} = \mathsf{mult}_u \restriction_{\mathbf{N}}$. There are only finitely many positions $|u|$ for which such j does not exist. Next, as there is no bound on j, we remove from $w[1, j - |u|]$ all cycles that do not contain neither of positions $j_0, \ldots, j_n$. The resulting word v has the length bounded by $(|Q_s| + 2) \cdot \mathsf{conf}(\mathbb{A}) = \mathbf{N}$ as $n \leq |Q_s|$. It follows that for every $q \in A$ we have $\mathsf{mult}_{uv}(q) \leq \mathbf{N}$ and $\mathsf{mult}_{uv}(q) \leq \mathsf{mult}_u(q) \restriction_{\mathbf{N}}$. Indeed, since cycle removal does not increase the multiplicity, for every $q \in A$ we have $\mathsf{mult}_{uv}(q) \leq \mathsf{mult}_{uw[1,j]}(q)$ and $\mathsf{mult}_{uw[1,j]} \restriction_{\mathbf{N}} = \mathsf{mult}_u \restriction_{\mathbf{N}}$. We show that the partial sum of weights of the master automaton at the position $|uv|$ in uvw is smaller than the partial sum at the position $|u|$ is uw, which implies that the transformation $uw \to uvw$, removes a position violating our assumption, and even applied infinitely many times, preserves optimality of the resulting words.

Let val be a function with $\mathrm{dom}(\mathsf{val}) = \mathrm{dom}(\mathsf{mult}_u)$ such that $\mathsf{val}(q)$ is the value accumulated in w (past the position $|u|$ i uw) by any slave automaton that is in the state q at the position $|u|$. Equivalently, that is the value accumulated by the same automaton past the position $|uv|$ in uvw. We call a slave automaton *active* if $\mathsf{val}(q) \geq 4 \cdot \mathbf{N}$, where q is the state of that automaton at the position $|u|$ (resp. $|uv|$). The value of the partial sum up to the position $|u|$ in uw is the value of all slave automata started before $|u|$. It consists of $(1) + (2)$, where

– (1) is the value all inactive slave automata plus the value of active slave automata accumulated up to the position $|u|$, and
– (2) is the value accumulated in w by all active automata past the position $|u|$.

Observe that $(2) = \sum_{q \in A} \mathsf{val}(q) \cdot \mathsf{mult}_u(q)$, where A is the set of states of active slave automata at the position $|u|$. The value of the partial sum up to the position $|uv|$ in uvw consists of $(1)' + (2)' + (3)$, where

– $(1)'$ is the value all inactive slave automata plus the value of active slave automata accumulated up to the position $|u|$ does not exceed (1),
– $(2)'$ is the value accumulated by active automata in w past position $|uv|$, and
– (3) is the value accumulated by all active slave automata on the word v, i.e., between the positions $|u|$ and $|uv|$ in uvw.

Note that $(1)'$ is bounded by (1), (3) is bounded by $2 \cdot \mathbf{N} \cdot |v| \leq 2 \cdot \mathbf{N}^2$, and $(2)' = \sum_{q \in A} \mathsf{val}(q) \cdot \mathsf{mult}_{uv}(q)$. We claim that $(2) - (2)' > (3)$, which means that the partial sum at the position $|uv|$ in uvw is smaller than the partial sum at the position $|u|$ in uw. Indeed, $\sum_{q \in A} \mathsf{mult}_u(q) - \mathsf{mult}_{uv}(q') > 2 \cdot \mathbf{N} - \mathbf{N} = \mathbf{N}$ and for each $q \in A$, $\mathsf{val}(q) \geq 4 \cdot \mathbf{N}$, therefore $(2) - (2)'$ is at least $4 \cdot \mathbf{N}^2$, which is greater than (3).

It follows that aforementioned transformation, even applied infinitely many times, will not increase the value of the resulting word. Therefore, there is s_0 and an optimal word such that at each position $s > s_0$ at most $2 \cdot \mathbf{N}$ will accumulate value grater than $4 \cdot \mathbf{N}$. □

Proof (of Lemma 17). We transform the automaton $\mathbb{A}$ to an equivalent deterministic $(\text{LimAvg}; \text{Sum}^+)$-automaton $\mathbb{A}_0$, which has an optimal run such that at each position the number of running slave automata is bounded by $\mathbf{c}$. Due to Lemma 18 there is s_0 and an optimal word such that at each position $s > s_0$, at most $2 \cdot \mathbf{N}$ slave automata accumulate value grater than $4 \cdot \mathbf{N}$ past the position s. We call that word w_{opt}.

The define an automaton $\mathbb{A}_0$ by modifying $\mathbb{A}$ in two ways. First, we extend the input alphabet to include the marking of the position s_0 in w_{opt}. Prior to that marking, a modified automaton starts only dummy slave automata that immediately terminate. Past that marking is simulates $\mathbb{A}$ Second, we modify each slave automaton of $\mathbb{A}$ in such a way that is runs as long as it can accumulate the value exceeding $4 \cdot \mathbf{N}$. More precisely, the master automaton starts only automata that return values exceeding $4 \cdot \mathbf{N}$. For other slave automata it "guesses" their value from the set $\{0, \ldots, 4 \cdot \mathbf{N}\}$ and runs a dummy automaton that takes only a single transition of this weight. As it is deterministic, we assume that the "guess" is encoded in the input word. Started slave automata run as long as they can accumulate the value exceeding $4 \cdot \mathbf{N}$. Once a slave automaton guesses that this is not possible, it takes a transition of the weight $4 \cdot \mathbf{N}$ and terminates. Again, that "guess" is encoded in the input word, therefore the master automaton is able to verify that this "guess" is correct.

The automaton $\mathbb{A}_0$ simulates the runs of $\mathbb{A}$ past the position s_0 and each running slave automaton accumulates the value exceeding $4 \cdot \mathbf{N}$. Therefore, there is a run of $\mathbb{A}_0$ on a word corresponding to w_{opt} such that at most $2 \cdot \mathbf{N} + 1$ slave automata runs concurrently. The automaton $\mathbb{A}_0$ is equivalent to $\mathbb{A}$, as the return values of slave automata pas the position s_0 are the same and the LimAvg value function does not depend on finite prefixes. □

Lemma 19. *Let $\mathbb{A}$ be a deterministic $(\text{LimAvg}; \text{Sum}^+)$-automaton that has an accepting run. There is an optimal run of $\mathbb{A}$ such that among every consecutive $\mathsf{conf}(\mathbb{A})$ steps, the master automaton of $\mathbb{A}$ takes a non-silent transition.*

Proof. Consider an optimal run of $\mathbb{A}$ on a word w and positions i, j such that $i + 2 \cdot \mathsf{conf}(\mathbb{A}) < j$ and $\mathbb{A}$ takes only silent transitions between i and j.

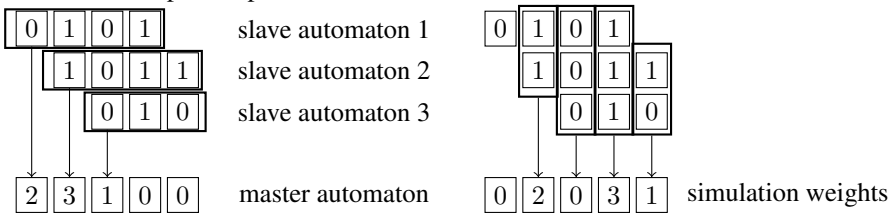
Observe that there are positions $i < i', j' < j$ with the same configuration. Consider a word w' resulting from removing $w[i', j']$ from w. The partial sum of the weights of the master automaton up to the position $j-(j'-i')$ on w' does not exceed the partial sum up to the position j on w. These partial sums are divided, in the average, by the same number of steps. Thus, the value of the word will not increase even if we can carry out this operation infinitely often. One should be careful not to remove all positions with accepting states. However, it is not a serous problem as we can insert sparsely subwords with an accepting state (after $1, 2, \ldots, 2^k, \ldots$ time increase steps). Such an operation will not increase the limit average of the run. □

Lemma 20. *The emptiness problem for* (LIMAVG; SUM$^+$)*-automata reduces to the emptiness problem for* sil(LIMAVG)*-automata.*

Proof. Let $\mathbb{A}$ be a (LIMAVG; SUM$^+$)-automaton. Due to Lemma 17 it can be transformed to an equivalent automaton $\mathbb{A}'$ which has an optimal run of width bounded by $\mathbf{c}$, i.e., such that at each position the number of running slave automata is bounded by $\mathbf{c}$.

We define a sil(LIMAVG)-automaton $\mathcal{A}$ such that the emptiness problems for $\mathbb{A}'$ and $\mathcal{A}$ coincide. The automaton $\mathcal{A}$ that simulate runs of $\mathbb{A}'$ of width bounded by $\mathbf{c}$ works as follows. It keeps track of the current configuration and the multiplicity of $\mathbb{A}'$; it needs to check that consecutive configurations and multiplicities follow form the transitions of $\mathbb{A}'$ A reader can convince itself that such an automaton can be easily constructed. There is a bijective correspondence between runs of $\mathbb{A}'$ of width bounded by $\mathbf{c}$ and runs of $\mathcal{A}$. That is, given a run of $\mathcal{A}$ one can construct a run of $\mathbb{A}'$ of width bounded by $\mathbf{c}$, and vice versa. In particular, there is an optimal run of $\mathbb{A}'$ that can be simulated by $\mathcal{A}$.

However, there is a substantial difference in how $\mathbb{A}'$ and $\mathcal{A}$ aggregate their weights. In $\mathbb{A}'$ a slave automaton started at a position k computes its value and returns it as a weight of the transition at the position k, whereas in $\mathcal{A}$, a simulated slave automaton runs concurrently to the master automaton and it adds its weights to the value of $\mathcal{A}'$ at each step. One can imagine a matrix that stores at the position (i, j) the weight of the transition of j-th slave automaton at the position i. Then, the value of $\mathbb{A}'$ is the limit average of the sums of rows, whereas the value of $\mathcal{A}$ is the limit average of the sums of columns. Despite this difference, we show that the value of an optimal run of $\mathcal{A}$ and an optimal run of $\mathbb{A}'$ coincide, which implies that the emptiness problem for $\mathcal{A}$ and $\mathbb{A}'$ coincide.



Observe that for every run $(\Pi, \pi_1, \pi_2, \ldots)$ of $\mathbb{A}'$ with at most $\mathbf{c}$ concurrently running slave automata, and the corresponding simulation run η of $\mathcal{A}$ the following holds: for every k we have $\sum_{i=1}^{k}(C(\eta))[i] \leq \sum_{i=1}^{k}(C(\pi_i))$. Therefore, the value of the optimal run of $\mathcal{A}$ does not exceed the value of the optimal run of $\mathbb{A}'$.

Conversely, consider an optimal run η of $\mathcal{A}$. Since $\mathcal{A}$ is a sil(LIMAVG)-automaton, the partial limit average sums of η converge. Consider a sequence $\{a_i\}_{i\geq 0}$ with $a_0 = 4$ and $a_{i+1} = a_i + \log a_i$. Observe that $|\{a_i : a_i < k\}| = o(k)$, i.e., $\lim_{k\to}\infty\frac{|\{a_i:a_i<k\}|}{k} = 0$. Let η' be a run obtained from η by injecting reset words on positions a_i in η, i.e., the words

that terminate all slave automata that are currently active. Such words exist and their length is bounded by $|Q_s| \cdot \mathsf{conf}(\mathbb{A})$. The value of η' is the same as the value of η. Indeed, for every $k > 0$ we have

$$\sum_{i=1}^{k}(C(\eta))[i] \leq \sum_{i=1}^{k}(C(\eta'))[i] \leq (\sum_{i=1}^{k}(C(\eta))[i] + o(k))$$

The first inequality is clear and the second follows from the fact that there are $|\{a_i : a_i < k\}| = o(k)$ reset words up to the position k and the cost contribution of each of them is bounded by the product of (1),(2),(3), where (1) is the maximal length of a reset word, (2) is the number of currently running slave automata, and (3) is the maximal weight a slave automaton can take. Note that (1),(2),(3) are bounded by a constant, hence the total contribution of inserted reset words up to the position k is $o(k)$. Due to Lemma 19, there are at least $\frac{k}{\mathsf{conf}(\mathbb{A})}$ non-silent transitions up to position k. Hence the limit averages of η and η' are equal, i.e., η' is an optimal run of $\mathcal{A}'$ as well.

Now, we consider a run of $(\Pi, \pi_1, \pi_2, \ldots)$ of $\mathbb{A}'$ that corresponds to η'. Observe that each slave automaton started at the position k, terminates after at most $\log k$ steps. Therefore, we have that the partial sum of weights of $(\Pi, \pi_1, \pi_2, \ldots)$ up to k is bounded by the partial sum of weights in η' up to $k + \log k$, i.e., for every k,

$$\sum_{i=1}^{k}(C(\eta'))[i] \leq \sum_{i=1}^{k}(C(\pi_i)) \leq \sum_{i=1}^{k+\log k}(C(\eta'))[i]$$

However, each $(C(\eta'))[i]$ is bounded by a constant, the maximal weight of slave automata in $\mathbb{A}$ times the number of slave automata. Therefore, $\sum_{i=k+1}^{k+\log k}(C(\eta'))[i] = O(\log k)$. Again, since the number of non-silent transitions up to the position k is $O(k)$, $\lim_{k\to\infty}\frac{O(\log k)}{O(k)} = 0$ and the limit averages of $(C(\eta'))[1], (C(\eta'))[2], \ldots$ and $C(\pi_1), C(\pi_2), \ldots$ are equal. Thus, the value of an optimal run of $\mathbb{A}'$ does not exceed the value of an optimal run of $\mathcal{A}$. In consequence, the values of optimal runs of $\mathcal{A}$, $\mathbb{A}'$ and $\mathbb{A}$ coincide.

□

F The universality problem for functional (LIMAVG; SUM$^+$)-automata.

Universality, in contrast to emptiness, is a two player game. The maximizer, whose objective is to maximize the value of the resulting run, chooses the letters and minimizer chooses the transition of the automaton. That game is a game with imperfect information (blind game for the maximizer), i.e., the choices of maximizer do not depend on the choice of the transitions of the minimizer. However, since the nested automaton is functional, the outcome of the game does not depend on the choices of the minimizer, provided that the resulting run is accepting. Therefore, we can assume that the maximizer chooses letters and transitions and its objective is to construct an accepting run of the maximal value.

Lemma 21. *Let $\mathbb{A}$ be a functional* (LIMAVG; SUM$^+$)*-automaton whose all slave automata have weights $\{0, 1\}$. Then, one of the following holds:*

1. *For every accepting run, there is a position s_0 such that every slave automaton started after s_0 accumulates the value not exceeding $\mathsf{conf}(\mathbb{A})$.*

2. *The automaton $\mathbb{A}$ has an accepting run of infinite value (whose value exceeds every $\lambda > 0$).*

Proof. Assume that (1) does not hold. Then, there is an accepting run such that some slave automaton returns values that exceed the value $\mathsf{conf}(\mathbb{A})$ infinitely often. Observe that if a slave automaton $\mathfrak{B}$ accumulates a value exceeding $\mathsf{conf}(\mathbb{A})$ during a run π, then the nested automaton $\mathbb{A}$ is in the same configuration at least twice during the run π and meanwhile $\mathfrak{B}$ increases its value. Therefore, one can pump the run of the nested automaton to increase the value returned by $\mathfrak{B}$. It follows that we can pump successively the run on $\mathbb{A}$ such that infinitely often the following holds: a slave automaton started at a position k accumulates the value exceeding k^2. A run with such a property has an infinite weight according to the semantics $\textsc{LimAvg}(\pi) = \limsup_{k\to\infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\pi))[i]$. □

Now, we are ready to prove decidability of the universality problem for functional $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata.

Proof. If (1) holds, $\mathbb{A}$ is equivalent to a functional $(\textsc{LimAvg}; \textsc{Sum}^B)$, where $B = \mathsf{conf}(\mathbb{A})$, which in turn is equivalent to a functional $\mathsf{sil}(\textsc{LimAvg})$-automaton. The universality problem is decidable for $\mathsf{sil}(\textsc{LimAvg})$-automata (Lemma 14). Otherwise, if (2) holds, then an answer to the universality problem for $\mathbb{A}$ is "No" for every λ. Now, it can be detected whether (1) or (2) holds by reduction to the universality problem for functional $(\textsc{LimSup}; \textsc{Sum}^+)$-automata. □

G The inclusion problem

The emptiness and universality problems reduce to the inclusion problem. Therefore, a class of nested weighted automata can have decidability for the inclusion problem only if both the emptiness and the universality problems are decidable.

Non-deterministic nested weighted automata. Hence, in the non-deterministic case, for value functions studied in Table 2, the inclusion problem can be decidable only in two cases:

1. for $(f; g)$-automata, where g is regular value function, and $f \in \mathsf{InfVal} \setminus \{\textsc{LimAvg}\}$;
2. for $(f; \textsc{Sum}^+)$-automata, where $f \in \{\textsc{Inf}, \textsc{LimInf}, \textsc{Sup}, \textsc{LimSup}\}$.

In fact, in (2) case the inclusion problem is undecidable as well. Indeed, inclusion of $\textsc{Sum}^+$-automata over finite words reduces to the inclusion of $(f; \textsc{Sum}^+)$-automata, where $f \in \{\textsc{Inf}, \textsc{LimInf}, \textsc{Sup}, \textsc{LimSup}\}$. It has been shown in [1] that the inclusion problem for $\textsc{Sum}^+$-automata is undecidable. Therefore, the inclusion problem in (2) case is undecidable. As automata in (1) case are equivalent to f-automata for $f \in \{\textsc{Inf}, \textsc{LimInf}, \textsc{Sup}, \textsc{LimSup}\}$ (Lemma 13), the inclusion problem is decidable [8].

H Details of example from Section 5.2

Consider the model M for two processes communicating through a channel, where every sent packet is delivered in the next state. Let \$ denote the event of neither sending or receiving packets, s_1 and r_1 (resp. s_2 and r_2) the send and receive for process 1 (resp. process 2). The language of M can be described as a regular expression as follows: $((\$)^* \cdot (s_1 r_1)^* \cdot (\$)^* \cdot (s_2 r_2)^*)^\omega$. Note that d_M must assign value 0 to every trace in the

language of M. However d_M needs to assign values to traces where the delivery of packets can be delayed by a finite amount. Hence we first need to relax the language of M as M_R such that every packet sent is received with a finite delay; and d_M assigns values to traces in the language of M_R. The relaxed language M_R is obtained as follows: consider the following languages L_1 and L_2

$$L_1 = ((\$)^* \cdot (s_1(\$)^* r_1)^* \cdot (\$)^*)^\omega; \quad \text{and} \quad L_2 = ((\$)^* \cdot (s_2(\$)^* r_2)^* \cdot (\$)^*)^\omega;$$

where L_1 denotes that every sent for process 1 can be delayed by a finite amount and analogously L_2 for process 2. The language of M_R is the *shuffle* (arbitrary interleavings) of L_1 and L_2. The products of the automaton $\mathbb{A}_D$ defined in Section 5.2 along with an automaton for M_R gives the desired similarity measure for the example.