IST AUSTRIA

Institute of Science and Technology

Nested Weighted Automata

Krishnendu Chatterjee and Thomas A Henzinger and Jan Otop

Nested Weighted Automata *

Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop

IST Austria

April 23, 2015

Abstract

Recently there has been a significant effort to handle 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, some basic system properties such as average response time cannot be expressed using weighted automata, nor in any other know decidable formalism. In this work, we introduce nested weighted automata as a natural extension of weighted automata which makes it possible to express important quantitative properties such as average response time. In nested weighted automata, a master automaton spins off and collects results from weighted slave automata, each of which computes a quantity along a finite portion of an infinite word. Nested weighted automata can be viewed as the quantitative analogue of monitor automata, which are used in run-time verification. We establish an almost complete decidability picture for the basic decision problems about nested weighted automata, and illustrate their applicability in several domains. In particular, nested weighted automata can be used to decide average response time properties.

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. For example, quantitative specifications such as accumulated sum can express properties like number of a events between b events, or memory consumption, whereas long-run average can express properties related to reliability requirements such as average success rate.

For Boolean properties regular languages provide a robust specification framework. Finding the analogue of regular languages for quantitative specifications is an active research area [4, 12, 16]. Some of the key features of such a specification framework are (1) expressiveness, i.e., whether the formalism can express properties of interest; (2) ease of specification, i.e., whether the properties can be stated naturally; (3) computability, i.e., whether the basic decision problems can be solved —ideally with elementary complexity— for interesting fragments of the formalism; and (4) robustness, i.e., whether the formalism is robust against small changes in its definition.

While automata are an expressive, natural, elementarily decidable, and robust framework for expressing Boolean properties, weighted automata provide a natural and flexible framework for expressing quantitative[1] properties [12]. 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 aggregates 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 [16]. They have been extended to infinite words with limit averaging or supremum as value

---

[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.

function [9, 11, 12]. While weighted automata over semirings can express several quantitative properties [25], they cannot express the following basic quantitative properties.

Example 1. *Consider infinite words over $\{r, g, i\}$, where r represents requests, g represents grants, and i represents idle. A first basic property is the long-run average frequency of r's, which corresponds to the average workload of a system. A second interesting property is the average number of i's between a request and the corresponding grant, which represents the long-run average response time of the system.*

While weighted automata with limit-average as value function can express the average workload property (which weighted automata over semirings cannot express), perhaps surprisingly, they are not capable of expressing the long-run average response time. 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-run average response time can be unbounded. However, 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 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 the case of Boolean finite automata, nested automata are no more expressive than their non-nested counterpart, we show that 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 in the following example.

Example 2. *In Example 1 there is only a single type of request and grant, but in general there can be multiple types of requests and grants, and the intervals between requests and corresponding grants for different requests may overlap. Using a nested weighted automaton, the average response time can be specified across all requests. We illustrate this for two types of requests and corresponding grants. The input alphabet is $\{r_1, g_1, r_2, g_2, i\}$. At every request r_1 (resp. r_2) the master automaton spins off a slave automaton $\mathfrak{B}_1$ (resp. $\mathfrak{B}_2$) with a sum value function, which counts the number of idle events to the next corresponding grant g_1 (resp. g_2). Observe that many slave automata may run concurrently. Indeed, for the word $(r_1^n r_2^n g_1 g_2)^\omega$, at all positions with the letter g_1 there are $2 \cdot n$ slave automata that run concurrently. The master automaton with limit-average value function then averages the response times returned by the slave automata.*

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 and complexity 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 the 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 includes the long-run average response time property, then the emptiness question is decidable in exponential space. Along with the decidability results, we also establish parametric complexity results, that show that when the total size of the slave automata is bounded by constant (which is the case for average response property), then for all decidability results the complexity matches that of Boolean non-nested automata (see Remark 23). 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 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 (in Table 3 and Table 4 for parametric complexity results) in Section 4.4.

Third, nested weighted automata provide a convenient formalism to express quantitative properties. In the Boolean case, *monitor automata* offer a natural compositional way of specifying complex temporal properties [26], and nested weighted automata can be seen as a quantitative extension of monitor automata. Each monitor automaton tracks a subproperty (which corresponds to a slave automaton in our formalism), and the results of the monitor automata are combined by another monitor automaton (which corresponds to the master automaton in our formalism).

- A key advantage of the monitor-automaton approach is that it allows complex specifications to be decomposed into subproperties, which eases the task of specification. Our nested weighted automata enjoy the same advantage: e.g., for long-run average response time, each slave automaton computes the response time (as a sum automaton) of an event, and the master automaton averages the response times. Formally, we show that deterministic nested weighted automata can be exponentially more succinct than non-deterministic weighted automata even when they express the same property (Theorem 25). Moreover, monitor automata are used heavily in run-time verification [20]. 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 on-the-fly an approximation of the value.

- More importantly, for Boolean properties monitor automata simply provide a more convenient framework for specification, as they are equally expressive as standard automata, whereas we show that nested weighted automata are strictly more expressive than non-nested weighted automata (e.g., long-run average response time, which cannot be expressed using non-nested weighted automata, can be expressed using nested weighted automata.

In other words, we provide a natural combination of weighted automata (for quantitative properties) and nesting of automata (for ease of expressiveness), and as a result obtain a more expressive, elementarily decidable, and convenient quantitative specification framework.

Finally, we illustrate the applicability of nested weighted automata in several domains. (1) We show that the *model-measuring* problem of [21] 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. (2) 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 that 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.

In summary, we introduce nested weighted automata, which offer an expressive and convenient quantitative specification framework, and establish that the basic verification problems are decidable for several interesting fragments (which include the long-run average response time property). While there exist many frameworks to express quantitative properties (that we discuss in details in Section 7), there exists no framework (to the best of our knowledge) that can express the average response time property and admit algorithms with elementary time complexity for the basic decision problems. We present a framework (of nested weighted automata) that can express such basic system properties and have decidable algorithms with elementary complexity.

The paper is a full version of [13].

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 and $i, j \in \mathbb{N}$, we define $w[i]$ as the i-th letter of w and $w[i, j]$ as the word $w[i]w[i+1]\ldots w[j]$. 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* $\pi = \pi[0]\pi[1]\ldots$ is a sequence of states such that $\pi[0] \in Q_0$ and for every $i \in \{1, \ldots, |w|\}$ 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 $j+1$ is *accepting* if $\pi[j] \in F$; and an infinite run π is *accepting*, if there exist infinitely many j such that $\pi[j] \in F$. Let $\mathsf{Acc}(w) \subseteq \mathsf{Run}(w)$ denote the set of accepting runs. A word w is accepted iff $\mathsf{Acc}(w)$ 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 Γ-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 Γ-labeled automaton, where Γ is a finite subset 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 non-empty 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 $f(\pi)$ 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 non-empty 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, i.e., the value of a word that has no accepting runs is infinite. Every run π on an empty word has length 1 and the sequence $C(\pi)$ is empty, hence we define the value $f(\pi)$ as an external (not a real number) value $\bot$. Thus, the value of the empty word is either $\bot$, if the empty word is accepted by $\mathcal{A}$, or ∞ otherwise. 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 natural variants for value functions. For finite runs we consider the following value functions: for runs of length $n+1$ we have

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

We denote the above class of value functions for finite words as $\mathsf{FinVal} = \{\mathrm{MAX}, \mathrm{MIN}, \mathrm{SUM}, \mathrm{SUM}^{+}, \mathrm{SUM}^{B}\}$.

Although, the absolute sum value function SUM^{+} can be equivalently expressed by SUM restricted to the class of weighted automata with non-negative weights, we consider SUM and SUM^{+} separately, as the resulting automata differ in complexity results.

For infinite runs we consider:

1. *Supremum and Infimum, and Limit supremum and Limit infimum:*
   - $\mathrm{SUP}(\pi) = \sup\{(C(\pi))[i] : i > 0\}$,
   - $\mathrm{INF}(\pi) = \inf\{(C(\pi))[i] : i > 0\}$,
   - $\mathrm{LIMSUP}(\pi) = \limsup\{(C(\pi))[i] : i > 0\}$, and
   - $\mathrm{LIMINF}(\pi) = \liminf\{(C(\pi))[i] : i > 0\}$.

2. *Limit average:* $\mathrm{LIMAVG}(\pi) = \limsup_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k}(C(\pi))[i]$.

We denote the above class of value functions for infinite words as $\mathsf{InfVal} = \{\mathrm{SUP}, \mathrm{INF}, \mathrm{LIMSUP}, \mathrm{LIMINF}, \mathrm{LIMAVG}\}$.

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 non-empty word w such that $\mathcal{L}_\mathcal{A}(w) \leq \lambda$ (resp. for all non-empty 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 3. *(1) The emptiness problem is decidable in polynomial time for all value functions we consider [18, 25]. (2) The universality problem is undecidable for SUM-automata with $\{-1, 0, 1\}$ weights and LIMAVG-automata with $\{0, 1\}$ weights; and decidable in polynomial space for all other value functions [1, 15, 8, 23]. (3) The universality problem is decidable for all value functions for deterministic and functional automata [19].*

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 a 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 a usual automaton, except that before it takes a transition, it starts 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 runs on some finite part of the input word. Once a slave automaton terminates, 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 slave automaton runs on the empty word and returns $\bot$; we refer to such transitions as *silent* transitions. A given run of a nested weighted automaton, which consists of a run of the master automaton and runs of slave automata, is accepting if it consists of accepting runs only. Finally, the value of an accepting 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).

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 a $\{1, \ldots, k\}$-labeled automaton over infinite words (where labels are indices of slave automata), called the *master automaton*, $f \in \mathsf{InfVal}$ 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) for every $i > 0$ we have π_i is a run of the automaton $\mathfrak{B}_{C(\Pi[i-1], w[i], \Pi[i])}$, referenced by the label $C(\Pi[i-1], w[i], \Pi[i])$ of the master automaton, on some finite subword $w[i, j]$ of w. 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) and infinitely many runs of slave automata have length greater than 1 (the master automaton takes infinitely many non-silent transitions). The value of the run $(\Pi, \pi_1, \pi_2, \ldots)$ is defined as $\mathsf{sil}(f)(v(\pi_1)v(\pi_2)\ldots)$, where $v(\pi_i)$ is the value of the run π_i in the corresponding slave automaton and $\mathsf{sil}(f)$ is the value function that applies f on sequences after removing $\bot$ symbols. 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. We require accepting runs to contain infinitely many non-silent transitions because f is a value function over infinite sequences, so we need the sequence $v(\pi_1)v(\pi_2)\ldots$ with $\bot$ symbols removed to be infinite.

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 examples.

Example 4 (Stuttering). *Consider the nested weighted automaton $\mathbb{A}^1_{stu} = \langle \mathcal{A}^1_{mas}; \mathrm{LIMAVG}; \mathfrak{B}_1, \mathfrak{B}_2 \rangle$ where each slave automaton is a 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 $\mathfrak{B}_1$ accepts words from a^*b and assigns to a word $a^k b$ value k. The automaton $\mathfrak{B}_2$ accepts words from b^*a and assigns to a word $b^k a$ value k.*

Consider a word $(aaab)^\omega$. A run of $\mathbb{A}^1_{stu}$ on $(aaab)^\omega$ is depicted in Figure 1. The value of the word is $\frac{7}{4}$. Note that $\mathbb{A}^1_{stu}$ accepts only words with infinite number of a's and b's, as otherwise, some slave automaton would not terminate. For word $w = (a^n b)^\omega$ the value is $(\frac{(n+1)n}{2} + 1) \cdot \frac{1}{n+1}$, 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 the automaton $\mathbb{A}^2_{stu} = \langle \mathcal{A}^2_{mas}; \mathrm{LIMAVG}; \mathfrak{B}_1, \mathfrak{B}_2, \mathfrak{B}_3 \rangle$, where $\mathfrak{B}_3$ has only a single

state, which is accepting, and it has no transitions. Thus, $\mathfrak{B}_3$ accepts on the empty word and invoking it is a way for $\mathcal{A}^2_{mas}$ to take a silent transition. Intuitively, each slave automaton counts how many times a given letter occurs. Thus the value computed by the nested weighted automaton is the average letter repetition (or average stuttering). Silent transitions, produced by calling the automaton $\mathfrak{B}_3$, enable $\mathcal{A}^2_{mas}$ to compute average only over positions where a new block starts.

Example 5 (Average response time). *Consider the specification for average response time defined as follows: we consider words for the alphabet $\{r, g, i\}$, where r denotes a request, g denotes a grant, and i denotes idle (no request or grant). Consider a word w, and a position j, such that $w[j]$ is a request, and then the response time in position j is the distance to the closest grant, i.e., the response time is $j' - j$ where $j' > j$ is the least number greater than j 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 i takes a silent transition (i.e., it is a single state automaton with r labeled as 1, and g and i labeled as 2). The slave automaton $\mathfrak{B}_1$ counts the number of steps till the first g and the slave automaton $\mathfrak{B}_2$ accepts only the empty word, which is used to produce silent transitions. The nested weighted automaton specifies the average response time property. As discussed in Section 1 since the average response time can be unbounded, it cannot be expressed by a non-nested limit-average automaton, whose value is bounded by the maximal weight that occurs in it.*

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 weighted 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.

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

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 and complexity 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 3 it follows that the universality problem for $(f; g)$-automata is decidable only if the universality problem is decidable for f-automata and g-automata.

Theorem 6. *(1) For $f \in$ InfVal, the universality problem for $(f; \text{SUM})$-automata is undecidable. (2) For $g \in$ FinVal, the universality problem $(\text{LIMAVG}; g)$-automata is undecidable.*
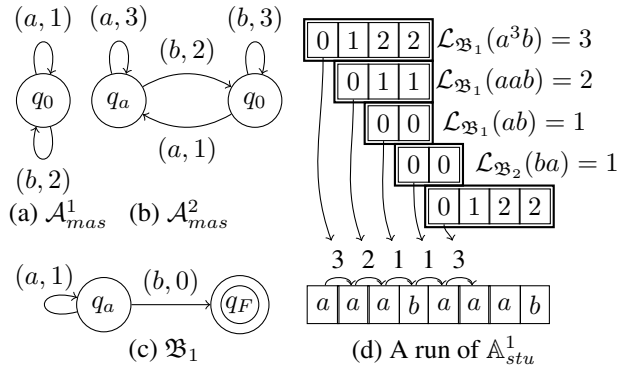
Proof of (1) from Theorem 6. We show a reduction of the universality problem for SUM-automata with weights $\{-1, 0, 1\}$, which is undecidable (Theorem 3), to the universality problem for $(f; \text{SUM})$-automata, where $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$. The case $f = \text{LIMAVG}$ 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 $w_1 \# w_2 \# \dots$. 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 $w_1 \# w_2 \# \dots$ is equal to the infimum of values $\mathcal{L}_\mathcal{A}(w_i)$. In particular, $\mathcal{L}_\mathbb{A}((w\#)^\omega) = \mathcal{L}_\mathcal{A}(w)$. Since for every word $w_1 \# w_2 \# \dots$ there exists i such that $\mathcal{L}_\mathbb{A}(w_1 \# w_2 \# \dots) = \mathcal{L}_\mathbb{A}((w_i \#)^\omega)$, 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). $\qquad\square$

Proof of (2) from Theorem 6. 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 3). Therefore, the universality problem for $(\text{LIMAVG}; g)$-automata is undecidable as well. $\qquad\square$

In the decidable cases, the lower bound for the emptiness and the universality problems is PSPACE. Recall, the finite automata intersection problem, which asks, given a set of deterministic finite automata, is there a finite word accepted by all of them, is PSPACE-complete [22]. That problem can be reduced to the emptiness problem of deterministic nested weighted automata. This implies PSPACE-hardness of the emptiness problem for deterministic nested weighted automata. Moreover, for deterministic nested weighted automata, the emptiness problem can be reduced to the universality problem. Therefore, the universality problem for deterministic nested weighted automata is also PSPACE-hard.

Proposition 7. *For all $f \in \mathsf{InfVal}$ and $g \in \mathsf{FinVal}$, the emptiness (resp. the universality) problem for deterministic nested weighted automata is* PSPACE-*hard.*

Proof. Let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be deterministic finite automata over the alphabet Σ. First, we modify each of them to obtain $\mathcal{A}_1^*, \dots, \mathcal{A}_n^*$ over $\Sigma \cup \{\#, \$\}$ such that $\#, \$ \notin \Sigma$ and for every $i \in \{1, \dots, n\}$ we have $\mathcal{A}_i^*$ accepts u iff $u = \#^k w \$$ and $\mathcal{A}_i$ accepts w. Observe that $\mathcal{A}_1^*, \dots, \mathcal{A}_n^*$ recognize prefix-free languages. Then, we define a nested weighted automaton whose slave automata are $\mathcal{A}_1^*, \dots, \mathcal{A}_n^*$ and the master automaton recognizes the language $(\#^n \{a, b\}^* \$)^\omega$. The master automaton invokes all slave automata on successive $\#$ letters. Thus, for a word $\#^n w_1 \$ \#^n w_2 \$ \dots$ to be accepted, the slave automaton $\mathcal{A}_1^*$ has to accept the words $\#^n w_1 \$, \#^n w_2 \$, \dots$, $\mathcal{A}_2^*$ has to accept the words $\#^{n-1} w_1 \$, \#^{n-1} w_2 \$, \dots$, and so on. It follows that the nested weighted automaton accepts the language $\#^n w_1 \$ \#^n w_2 \$ \dots$ such that all words w_1, w_2 are accepted by all automata $\mathcal{A}_1, \dots, \mathcal{A}_n$. Therefore, the nested weighted automaton accepts any word iff there exists a finite word accepted by all automata $\mathcal{A}_1, \dots, \mathcal{A}_n$. The nested automaton defined as above is deterministic. $\qquad\square$

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 8 (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 $\{q_1, \dots, q_n\} \subseteq \mathbb{Q}$ and there are regular languages $\mathcal{L}_1, \dots, \mathcal{L}_n$ such that*
 (i) every word accepted by $\mathcal{A}$ belongs to $\bigcup_{1 \le i \le n} \mathcal{L}_i$, and
 (ii) for every $w \in \mathcal{L}_i$, each run of $\mathcal{A}$ on w has the weight q_i.

Remark 9. *Regular weighted automata define the class of are equivalent to recognizable step functions [16]. However, we (implicitly) require regular languages $\mathcal{L}_1, \dots, \mathcal{L}_n$ to be disjoint, whereas the value of a recognizable step function at a word w is defined as the minimum $q_i \in \{q_1, \dots, q_n\}$ among such i's that w belongs to $\mathcal{L}_i$.*

We define the *description size* of a given regular weighted automaton $\mathcal{A}$, as the size of automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ recognizing languages $\mathcal{L}_1, \dots, \mathcal{L}_n$ that witness $\mathcal{A}$ being a regular weighted automaton.

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 $\textsc{Min}, \textsc{Max}, \textsc{Sum}^B$. Observe that the description size of a $\textsc{Min}$, $\textsc{Max}$ and $\textsc{Sum}^B$-automaton $\mathcal{A}$ is polynomial in $|\mathcal{A}|$, but it is exponential in the length of binary representation of B, for a $\textsc{Sum}^B$-automaton $\mathcal{A}$.

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 weighted 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, recall that $\mathsf{sil}(f)$ is the value function that applies f on sequences after removing silent transitions. The following Lemma 10 along with Theorem 3 implies Theorem 13.

Lemma 10 (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 a $\mathsf{sil}(f)$-automaton $\mathcal{A}$ (weighted automaton), that can be constructed in polynomial space, 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 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 $F_1 = F_m \times (\{-n, \dots, n\} \cup \{\bot\}) \times 2^{Q_S} \times 2^{Q_S}$ and $F_2 = Q_m \times (\{-n, \dots, n\} \cup \{\bot\}) \times 2^{Q_S} \times \emptyset$. 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.

The correctness follows from the construction. $\qquad\square$

The automaton $\mathcal{A}$ in Lemma 10 is constructed in polynomial space, which means that $\mathcal{A}$ can be represented implicitly, i.e., its exponential-size set of states is represented in a compact way and for each transition triple (q, a, q') one can compute in polynomial time whether that triple is a transition of $\mathcal{A}$ and what is its weight.

Remark 11. *The automaton $\mathcal{A}$ from Lemma 10 has exponential size in $|\mathbb{A}|$. More precisely, the size of $\mathcal{A}$ is exponential in the total size of slave automata of $\mathbb{A}$, but only polynomial in the size of the master automaton of $\mathbb{A}$.*

Proof. The set of states of the automaton $\mathcal{A}$ from is $Q = Q_m \times (\{-n, \dots, n\} \cup \{\bot\}) \times 2^{Q_S} \times 2^{Q_S}$, i.e., it is linear in the size of the master automaton of $\mathbb{A}$. Moreover, the weights of $\mathcal{A}$ are bounded by a constant n. Thus, $\mathcal{A}$ is polynomial in the size of the master automaton of $\mathbb{A}$. $\qquad\square$

Now, we show a simple lemma regarding weighted automata with silent moves.

Lemma 12. *Let $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$. (1) The emptiness problem for $\text{sil}(f)$-automata is in NLOGSPACE. (2) The universality problem for $\text{sil}(f)$-automata is in PSPACE.*

Proof. Given a $\text{sil}(f)$-automaton $\mathcal{A}$, where $f \in \text{InfVal}$, 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 $\text{sil}(\text{INF})$-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 $\text{sil}(\text{SUP})$-automaton $\mathcal{A}$ and its variant $\mathcal{A}^{(\lambda-1)}$. Thus, the emptiness and universality problems for $\text{sil}(\text{INF})$-automata (resp. $\text{sil}(\text{SUP})$-automata) and INF-automata (resp. SUP-automata) coincide. Now, a run of a $\text{sil}(\text{INF})$-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. As the emptiness problem for f-automata is in NLOGSPACE we have (1). The universality problem for f-automata is in PSPACE, hence we have (2). $\qquad\square$

Finally, we are ready to prove theorem characterizing complexity of decision problem for newsted weighted automata whose slave automata are $\{\text{MIN}, \text{MAX}, \text{SUM}^B\}$.

Theorem 13. *Let $g \in \{\text{MIN}, \text{MAX}, \text{SUM}^B\}$. The following assertions hold: (1) Let $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$. The emptiness problem for non-deterministic $(f; g)$-automata is PSPACE-complete. The universality problem for non-deterministic $(f; g)$-automata is PSPACE-hard and in EXPSPACE. (2) The emptiness problem for non-deterministic $(\text{LIMAVG}; g)$-automata is PSPACE-complete (3) The universality problem for functional $(\text{LIMAVG}; g)$-automata is PSPACE-complete.*

Proof. PSPACE-hardness in (1), (2) and (3) follows from Proposition 7. We will discuss containment separately:

(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 10 every $(f; g)$-automaton $\mathcal{A}$ is equivalent to some $\text{sil}(f)$-automaton $\mathcal{A}'$ of exponential size in $|\mathcal{A}|$. By Lemma 12, the emptiness problem for $\text{sil}(f)$-automata is in NLOGSPACE. The construction from Lemma 10 implies that the automaton $\mathcal{A}'$ can be represented implicitly, i.e., given two states q, q' the existence and weight of the transition (q, a, q') can be decided in polynomial time. Therefore, the emptiness problem for $(f; g)$-automata is in PSPACE.

By Lemma 12, and the universality problem for $\text{sil}(f)$-automata is in PSPACE and $|\mathcal{A}'|$ is of exponential size in $|\mathcal{A}|$, hence we have the universality problem for $(f; g)$-automata is in EXPSPACE.

(2): Lemma 10 state that $(\textsc{LimAvg}; g)$-automata are equivalent to $\mathsf{sil}(\textsc{LimAvg})$-automata, which enjoy decidability of the emptiness problem (Lemma 35) in NLogSpace. As in *(1)*, the automaton $\mathcal{A}'$ can be represented implicitly, hence the emptiness problem for $(\textsc{LimAvg}; g)$-automata is in PSpace.

(3): The universality problem for functional $(\textsc{LimAvg}; g)$-automata reduces to the emptiness problem for functional $(\textsc{LimAvg}; g)$-automata. It suffices to (1) first check that every word has an accepting run, which can be done in polynomial space, (2) construct a $(\textsc{LimAvg}; g)$-automaton $\mathbb{A}'$ by taking additive inverses of all weights in all slave automata of a given nested weighted automaton. The automaton $\mathbb{A}$ satisfies the universality problem with threshold λ iff it satisfies (1) and the automaton $\mathbb{A}'$ from (2) does not satisfy the emptiness problem with threshold $-\lambda$. Therefore, the universality problem for functional $(\textsc{LimAvg}; g)$-automata is in PSpace. $\qquad\square$

Remark 14. *Assume that the total size of slave automata is bounded. Then, by Remark 11, the size of the automaton $\mathcal{A}$ is polynomial in the size of the master automaton of a given nested automaton. In consequence, the emptiness problem for automata from (1) and (2) from Theorem 13 becomes* PTime *and the universality problem for automata from (1) from Theorem 13 becomes* PSpace-*complete.*

Theorem 13 covers the case for all classes of slave automata other than Sum- and $\textsc{Sum}^+$-automata, which we consider in the following two subsections.

4.2 Undecidability Results for Slave Sum Automata

In this section we study $(f; \textsc{Sum})$-automata and we present a crucial negative result.

Note that for weighted automata with the value function from FinVal or InfVal, the emptiness problem is decidable (for non-deterministic automata); and all decision problems are decidable for deterministic automata. In sharp contrast we establish that for deterministic $(\textsc{Sup}; \textsc{Sum})$-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 c_0, c_1 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 c_0, c_1, as well as zero and positive test, and for each counter we have three slave automata. For positions $i < j$, let c_0-*balance* (resp. c_1-balance) between position i and j denote the difference in the number of increments and decrements of the counter c_0 (resp. c_1) 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 c_0-balance and the opposite (the additive inverse) of c_0-balance between two consecutive zero tests. Given the zero test of the current position is satisfied, both automata return 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 c_0-balance plus 1 between the current position and the next zero test of c_0. The c_0-balance plus 1 does not exceed zero iff the value of c_0 at the current position is positive. We repeat a similar construction for c_1. The construction of slave automata does not depend on the given two-counter machine, therefore the reduction works even in the presence of a constant bound on the size of slave automata.

This establishes the undecidability for emptiness of $(\textsc{Sup}; \textsc{Sum})$-automata, and the proof also holds for $(\textsc{LimSup}; \textsc{Sum})$-automata. Also observe that since we establish the result for deterministic automata, we can take opposites of weights and change Sup (resp. LimSup) to Inf (resp. LimInf) and the emptiness problem to the universality problem.

Theorem 15 (Crucial undecidability result)**.** *(1) The emptiness problem for deterministic $(\textsc{Sup}; \textsc{Sum})$- and $(\textsc{LimSup}; \textsc{Sum})$-automata is undecidable. (2) The universality problem for deterministic $(\textsc{Inf}; \textsc{Sum})$- and $(\textsc{LimInf}; \textsc{Sum})$-automata is undecidable.*

Proof of (1) from Theorem 15. Given a Minsky machine $\mathcal{M}$, we construct a deterministic $(\textsc{Sup}; \textsc{Sum})$-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 a valid accepting computation of $\mathcal{M}$. As the problem, given a Minsky machine, does it have an accepting computation is undecidable, we conclude that the emptiness problem for deterministic $(\textsc{Sup}; \textsc{Sum})$-automata (resp. $(\textsc{LimSup}; \textsc{Sum})$-automata) is undecidable.

A Minsky 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 $(q, s, t) \rightarrow (q', v_1, v_2)$, where $s \in \{c_1 = 0, c_1 > 0\}, t \in \{c_2 = 0, c_2 > 0\}$ and $v_1, v_2 \in \{-1, 0, 1\}$. E.g. $(q, c_1 = 0, c_2 > 0) \rightarrow (q', +1, -1)$ 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 machines, a *run* and a *computation*. A *run* of a Minsky machine $\mathcal{M}$ is a sequence $(q_0, 0, 0), (q_1, \alpha_1, \beta_1), \dots, (q_n, \alpha_n, \beta_n)$ such that for every $i < n$ there is a transition of $\mathcal{M}$ $(q_i, s, t) \rightarrow (q_{i+1}, v_1, v_2)$ 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 $Q \times \{c_1 = 0, c_1 > 0\} \times \{c_2 = 0, c_2 > 0\} \times \{-1, 0, 1\} \times \{-1, 0, 1\}$ called *configurations*. A computation $(q_0, c_1 = 0, c_2 = 0, 0, 0), (q_1, s_1, t_1, x_1, y_1), \dots, (q_n, c_1 = 0, c_2 = 0, x_n, y_n)$ is *valid* iff there is an accepting run $(q_0, 0, 0), \dots, (q_n, \alpha_n, \beta_n)$ such that for every $i \in \{0, \dots, n\}$, $\alpha_i = \sum_{j=0}^{i} x_j$ 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 c_1-*balance* (resp. c_2)-balance 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 c_1-balance (resp. c_2-balance) between positions 1 and i. Thus, for $p \in \{1, 2\}$, a zero test (non-zero test) of c_p at the position i is valid iff c_p-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 are two possible 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.

Let $p \in \{1, 2\}$. First, we check validity of zero tests on c_p. All zero tests on c_p are valid iff c_p-balance between any two consecutive zero tests is zero. To check that this holds, the nested weighted automaton starts at each position i with a zero test two deterministic slave Sum-automata: $\mathcal{A}_{c_1=0}^+, \mathcal{A}_{c_1=0}^-$. The automaton $\mathcal{A}_{c_1=0}^+$ computes c_p-balance between i and the next zero test of c_p; it increments (decrements) its value whenever c_p is incremented (decremented), and it terminates at the next zero test of c_p. The automaton $\mathcal{A}_{c_1=0}^-$ does the opposite, i.e., it computes the additive inverse of c_p-balance between i and the next zero test of c_p. The values of these automata are inverses of each other and the maximum of their values is the absolute value of c_p-balance. Hence, the maximum of their values is less-or-equal to zero iff c_p-balance between i and the next zero test of c_p is 0. Thus, the values of all slave automata $\mathcal{A}_{c_1=0}^+, \mathcal{A}_{c_1=0}^-$ are less-or-equal to zero if and only if all zero tests of c_p 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 $\mathcal{A}_{c_1>0}$ that first increments its value to 1 and then computes c_p-balance between i and the next zero test of c_p. The value of c_p at the position i is strictly greater than 0 iff c_p-balance between the position i and the next position at which c_p 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 weighted 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_p are valid. In the above construction up to four automata has to be started at any configuration, while nested weighted 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 $w_1 \# w_2 \# \dots$ 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 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. $\qquad\square$

Proof of (2) from Theorem 15. 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 weighted 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 $-x$. $\qquad\square$

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

We now establish the remaining decidability results, namely, for slave automata with SUM$^+$ value function, and emptiness for (INF; SUM)-automata and (LIMINF; SUM)-automata. In contrast to the reduction of Lemma 10, for example, (LIMAVG; SUM$^+$)-automaton cannot be reduced to weighted sil(LIMAVG)-automata (Example 5).

Intuitive proof ideas. For $(f; \text{SUM}^+)$-automata, for $f \in \text{InfVal} \setminus \{\text{LIMAVG}\}$, we show that the decision problems can be reduced to the bounded sum value function; and then derive the decidability results from Theorem 13. The reductions are polynomial in the size of the master automaton. For (INF; 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. The constructed automaton is exponential in the size of a nested automaton, but only polynomial in the size of the master automaton, i.e., if the total size of slave automata and the threshold are bounded by a constant. We summarize the results in the following theorem.

Theorem 16. *(1) For $f \in \{\text{INF}, \text{LIMINF}\}$, the emptiness problem for $(f; \text{SUM})$-automata is PSPACE-complete. (2) For $f \in \{\text{SUP}, \text{LIMSUP}\}$, the universality problem for functional $(f; \text{SUM})$-automata is PSPACE-complete. (3) For $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$, the emptiness problem for $(f; \text{SUM}^+)$-automata is PSPACE-complete, and the universality problem for $(f; \text{SUM}^+)$-automata is PSPACE-hard and in EXPSPACE.*

Proof of (1) from Theorem 16. PSPACE-hardness follows from Proposition 7. For containment in PSPACE, let $\mathbb{A} = \langle \mathcal{A}_{mas}; \text{INF}; \mathfrak{B}_1, \dots, \mathfrak{B}_k \rangle$ be a nested weighted automaton (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 $\Sigma \cup \{\#, 1, \dots k\}$ of the form $wiv\#u'\#u$, where $w, v, u', u \in \Sigma^*$ and $i \in \{1, \dots k\}$, 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 component $\mathcal{A}_1$, 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, $\mathcal{A}_2$, is a weighted one and it computes the value of $\mathfrak{B}_i$ on v. Clearly, the size of $\mathcal{A}_2$ is proportional to the size of $\mathfrak{B}_i$. 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. The construction of $\mathcal{A}_1$ is similar to the construction from Lemma 10, hence the emptiness problem for $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ can be solved in polynomial space w.r.t. $|\mathbb{A}|$.

Assume that $\mathbb{A}$ is a (LIMINF; SUM)-automaton. We carry out virtually the same construction of a SUM-automaton over finite words $\mathcal{A}$. The automaton $\mathcal{A}$ accepts words $wiv\#u$ such that $\mathfrak{B}_i$ accepts v and $\mathbb{A}$ has an accepting run on $w(vu)^\omega$ at which the slave automaton invoked at the positions $w, wvu, \dots, w(vu)^k, \dots$ is $\mathfrak{B}_i$. The value of $\mathcal{A}$ on an accepted word $wiv\#u$ is the value of $\mathfrak{B}_i$ on v. It follows that if $\mathcal{A}$ has a run of value λ on $wiv\#u$, $\mathbb{A}$ has a run of the value λ on $w(vu)^\omega$. Conversely, if $\mathbb{A}$ has a run of the value λ, there is a reachable state q of the master automaton $\mathcal{A}_{mas}$ of $\mathbb{A}$ and a slave automaton $\mathfrak{B}_i$ such that infinitely often $\mathcal{A}_{mas}$ in the state q invokes $\mathfrak{B}_i$ which returns the value λ. Thus, there are words v, u such that $\mathfrak{B}_i$ on v returns the value λ and $\mathcal{A}_{mas}$ upon reading vu returns to the state q. Moreover, there is a word w such that $\mathcal{A}_{mas}$ reaches q from the initial state upon reading w. Therefore, the value of $w(vu)^\omega$ in $\mathbb{A}$ is at most λ. Hence, the emptiness problems for $\mathbb{A}$ and $\mathcal{A}$ coincide. Similarly to the (INF, SUM$^+$) case, the emptiness problem for $\mathcal{A}$ can be solved in polynomial space w.r.t. $|\mathbb{A}|$. $\square$

Remark 17. *The construction of $\mathcal{A}_1$ is similar to the construction from Lemma 10, hence it is polynomial in the size of the master automaton. Therefore, the emptiness problem for (INF; SUM)-automata (resp. (LIMINF; SUM)-automata) is in PTIME provided that the total size of slave automata is bounded.*

Proof of (2) from Theorem 16. PSPACE-hardness follows from Proposition 7. The universality problem for functional (INF; SUM)-automata (resp. (LIMINF; SUM)-automata) reduces to the emptiness problem for functional (SUP; SUM)-automata (resp. (LIMSUP; SUM)-automata). It suffices to (1) first check that every word has an accepting run, which can be done in polynomial space, (2) construct an automaton (INF; SUM)-automaton (resp. (LIMINF; SUM)-automaton) $\mathbb{A}'$ by taking inverses of all weights in all slave automata of a given nested weighted automaton. The automaton $\mathbb{A}$ satisfies the universality problem with threshold λ iff it satisfies (1) and the automaton $\mathbb{A}'$ from (2) does not satisfy the emptiness problem with threshold $-\lambda$. Therefore, the universality problem for functional (INF; SUM)-automata (resp. (LIMINF; SUM)-automata) is in PSPACE. $\square$

Proof of (3) from Theorem 16. PSPACE-hardness follows from Proposition 7.

Let λ be the threshold given in the emptiness (resp. universality) problem. Consider a $(f; \text{SUM}^B)$-automaton $\mathbb{A}^\lambda$, where $B = \lambda + 1$, obtained from $\mathbb{A}$ by changing each slave SUM$^+$ automaton $\mathfrak{B}$ into SUM$^{\lambda+1}$-automaton $\mathfrak{B}^\lambda$. Basically, such a SUM$^{\lambda+1}$-automaton $\mathfrak{B}^\lambda$ simulates runs of SUM$^+$-automata by implementing a $\lambda + 1$-bounded

counter in its states $Q \times \{0, \ldots, \lambda+1\}$, where Q is the set of states of $\mathfrak{B}$. If $\mathfrak{B}$ accumulates the value above λ, the automaton $\mathfrak{B}^\lambda$ returns just $\lambda + 1$, regardless of the actual value accumulated by $\mathfrak{B}$. The automaton $\mathbb{A}^\lambda$ is polynomial in λ, which can be exponential in the input size. Observe that for $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$, for every word w, $\mathbb{A}$ has a run on w of the value not exceeding λ threshold iff $\mathbb{A}^\lambda$ has. It follows that the emptiness (resp. universality) problem for $(f; \text{SUM}^+)$-automata with threshold λ reduces to the emptiness (resp. universality) problem for $(f; \text{SUM}^{\lambda+1})$-automata. Since SUM^B is a regular value function, Lemma 10 states that for $f \in \{\text{INF}, \text{SUP}, \text{LIMINF}, \text{LIMSUP}\}$, a $(f; \text{SUM}^{\lambda+1})$-automaton $\mathbb{A}^\lambda$ is equivalent to a $\text{sil}(f)$-automaton $\mathcal{A}$. Therefore, the emptiness (resp. the universality) problem for $(f; \text{SUM}^+)$-automata reduces to the emptiness (the universality) problem for $\text{sil}(f)$-automata. However, by employing Lemma 10, we get $\mathcal{A}$ of the size exponential in $|\mathbb{A}^\lambda|$ and doubly-exponential in $|\mathbb{A}|$. We show that the second exponential blow-up can be avoided.

We show that there exists a $\text{sil}(f)$-automaton $\mathcal{A}^-$, equivalent to $\mathcal{A}$ of the exponential size in the input size.

Infimum case. Let $f \in \{\text{INF}, \text{LIMINF}\}$. Original $\text{sil}(f)$-automaton $\mathcal{A}$ simulates runs of all slave automata of $\mathbb{A}^\lambda$. The modified $\text{sil}(f)$-automaton $\mathcal{A}^-$ simulates only a single $\text{SUM}^{\lambda+1}$-automaton at the time, which is chosen non-deterministically. For remaining slave automata, only their non-weighted counterparts are simulated, i.e., SUM^+ automata from $\mathbb{A}$ with weights removed. Since f is infimum of limit-infimum value function, the automata $\mathcal{A}$ and $\mathcal{A}^-$ are equivalent. The cardinality of the set of states of $\mathcal{A}^-$ is $O(2^{|\mathbb{A}|} \cdot |\mathbb{A}| \cdot B)$. Therefore, the size of $\mathcal{A}^-$ is exponential in the input size.

Supremum case. Let $f \in \{\text{SUP}, \text{LIMSUP}\}$. Recall that the set of states of $\mathfrak{B}^\lambda$ is $Q \times \{0, \ldots, \lambda+1\}$, where Q is the set of states of $\mathfrak{B}$. We obtain $\mathcal{A}^-$ from $\mathcal{A}$, by imposing the following condition: (*) at every position k, if $\mathcal{A}^-$ simulates two runs π_i, π_j of $\mathfrak{B}^\lambda$ that have states (q, w_1) resp. (q, w_2) at position k, with $w_1 > w_2$, $\mathcal{A}^-$ discards the run π_j (the one that has the state (q, w_2)). Intuitively, the run π_j can be completed to an accepting run that accumulates lower value than π_i, thus simulating it is redundant. We argue that $\mathcal{A}$ and $\mathcal{A}^-$ are equivalent.

The $\mathcal{A}^-$ simulates only a subset of slave automata. Since its value function is $\text{sil}(\text{SUP})$ or $\text{sil}(\text{LIMSUP})$, for every word w, the value of $\mathcal{A}^-$ does not exceed the value of $\mathcal{A}$. Conversely, consider an accepting run $(\Pi, \pi_1, \pi_2, \ldots)$ of $\mathbb{A}^\lambda$ on w. We can modify runs of slave automata $\pi_1, \pi_2, \ldots$ so that the modified run $(\Pi, \pi_1', \pi_2', \ldots)$ satisfies the following condition (**): at every position k in w, if runs π_i, π_j have states (q, w_1), resp. (q, w_2) at the positions corresponding to k, then they accumulate the same value in the remaining part of of the run. One can achieve that by changing the suffix of the run that accumulates greater value to the suffix of the other run. Such an operation of substituting a prefix decreases the value, hence it can be executed finitely many times for each run, and it will not produce infinite runs of slave automata. Observe that the modified run is an accepting run of $\mathbb{A}^\lambda$ of the value not exceeding the value of $(\Pi, \pi_1, \pi_2, \ldots)$.

Now, observe that for a run of $\mathbb{A}^\lambda$ satisfying (**), if runs π_i, π_j have states (q, w_1), resp. (q, w_2) at the positions k in w, with $w_1 > w_2$, the value of π_i is greater than the value of π_j, and the run π_j can be discarded. Such an operation corresponds to the condition (*) imposed by $\mathcal{A}^-$. Therefore, the values of w assigned by $\mathbb{A}^\lambda, \mathcal{A}$ and $\mathcal{A}^-$ are equal.

The cardinality of the set of states of $\mathcal{A}^-$ is $O((|\mathbb{A}| \cdot B)^{|\mathbb{A}|})$, which is exponential in the input size.

The emptiness (resp. the universality) problem of a $(f; \text{SUM}^+)$-automaton $\mathbb{A}$ reduces to the emptiness (the universality) problem for $\text{sil}(f)$-automaton $\mathcal{A}^-$ of the exponential size in $|\mathbb{A}| + \log(\lambda)$. Hence, by Lemma 12, for $(f; \text{SUM}^+)$-automata, the emptiness problem is in PSPACE and the universality problem is in EXPSPACE. $\quad\square$

Remark 18. *Let $f \in \{\text{INF}, \text{LIMINF}, \text{SUP}, \text{LIMSUP}\}$. Assume that the threshold λ is given in unary. Then, for $(f; \text{SUM}^+)$-automata, the emptiness problem is in* PTIME *and the universality problem is* PSPACE-*complete.*

Proof. Let $f \in \{\text{INF}, \text{LIMINF}, \text{SUP}, \text{LIMSUP}\}$. Assuming that the threshold is given in unary and the total size of slave automata is bounded, the size of $\mathcal{A}^-$ is polynomial in the size of $\mathbb{A}$. Therefore, the emptiness (resp., the universality) problem for $(f; \text{SUM}^+)$-automata reduce to the emptiness (resp., the universality) problem for f-automata. The emptiness problem for f-automata is in PTIME and the universality problem is PSPACE-complete. Hence, the result follows. $\quad\square$

Finally, we establish decidability of the emptiness problem with limit-average master automaton and SUM^+-automata as slave automata. The key proof idea is to show that values of certain runs of $(\text{LIMAVG}; \text{SUM}^+)$-automata coincide with the values of non-nested limit-average automata, and those runs have values arbitrarily close to the infimum over values of all runs. This also allows us to show the decidability of the universality problem for functional $(\text{LIMAVG}; \text{SUM}^+)$-automata.

Theorem 19. *The emptiness problem for $(\text{LIMAVG}; \text{SUM}^+)$-automata is* PSPACE-*hard and in* EXPSPACE*; and the universality problem for functional $(\text{LIMAVG}; \text{SUM}^+)$-automata is* PSPACE-*hard and in* EXPSPACE.

We present the proof of part (1) from Theorem 15 in Section 6. In the following, we show the proof of part (2) from Theorem 15.

Observe that for a run of a functional nested weighted automaton, as long as the run is accepting, its value does not depend on the choices of transitions. Therefore, we will focus on the construction of an accepting run with the maximal value to compute the minimal threshold for the functionality problem.

Lemma 20. *Let $\mathbb{A}$ be a functional* (LIMAVG; SUM$^+$)*-automaton and let Λ be the value bounding weights in all slave automata of $\mathbb{A}$. 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 $\Lambda \cdot \mathsf{conf}(\mathbb{A})$.*

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 $\Lambda \cdot \mathsf{conf}(\mathbb{A})$ infinitely often. Observe that if a slave automaton $\mathfrak{B}$ accumulates a value exceeding $\Lambda \cdot \mathsf{conf}(\mathbb{A})$ during a run π, then the nested weighted 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 weighted 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 $\mathrm{LIMAVG}(\pi) = \limsup_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\pi))[i]$. $\square$

Now, we are ready to prove decidability of the universality problem for functional (LIMAVG; SUM$^+$)-automata.

Proof of (2) from Theorem 15. If (1) holds, $\mathbb{A}$ is equivalent to a functional (LIMAVG; SUM^B)-automaton $\mathbb{A}'$, where $B = \Lambda \cdot \mathsf{conf}(\mathbb{A})$. The size of $\mathbb{A}'$ is exponential in $|\mathbb{A}|$. The universality problem for functional (LIMAVG; SUM^B)-automata is PSPACE-complete, which implies the the universality problem for functional (LIMAVG; SUM$^+$)-automata is in EXPSPACE. 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 (LIMSUP; SUM$^+$)-automata, which is PSPACE-complete. $\square$

Remark 21. *The size of $\mathbb{A}'$ is polynomial in the size of the master automaton of $\mathbb{A}$. Therefore, the universaility problem is in* PSPACE. *The universality problem for functional* SUM$^+$*-weighted automata is* PSPACE*-hard, hence the universality problem for* (LIMAVG; SUM$^+$)*-automata is* PSPACE*-complete assuming that the total size of slave automata is bounded.*

4.4 Summary and Open problems

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 we present as a conjecture.

Conjecture 22. *The emptiness problem for non-deterministic* (LIMAVG; SUM)*-automata is decidable.*

Tables 1 and 2 summarize our results.

Complexity. The decision problems are PSPACE-complete, in EXPSPACE, or undecidable. We show in Theorem 25 that (deterministic) nested weighted automata are exponentially more succinct than (non-deterministic) weighted automata, which explains EXPSPACE complexity of some universality problems.

We present the proof of the emptiness case from Theorem 19 in Section 6.

Discussion on inclusion. 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 $\mathcal{L}_{\mathbb{A}_1}(w) \leq \mathcal{L}_{\mathbb{A}_2}(w)$. Therefore, for decidability of the inclusion problem both the emptiness and the universality problem must be decidable. 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 $f \in \mathsf{InfVal} \setminus \{\mathrm{LIMAVG}\}$;

2. for $(f; \mathrm{SUM}^+)$-automata, where $f \in \mathsf{InfVal} \setminus \{\mathrm{LIMAVG}\}$.

		INF / LIMINF	SUP / LIMSUP	LIMAVG
MIN, MAX SUM^B	Empt.	PSP.-c (13)		
	Univ.			
SUM	Empt.	PSP.-c (16)	Undec. (15)	Open (22)
	Univ.	Undec. (15)	PSP.-c (16)	
SUM$^+$	Empt. Univ.	PSP.-c (13)		EXPSP. (19)

Table 1: Decidability and complexity 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. PSP. (resp. EXPSP.) denotes PSPACE (resp. EXPSPACE).

		INF / LIMINF	SUP / LIMSUP	LIMAVG
MIN, MAX SUM^B	Empt.	PSP.-c (13)		
	Univ.	EXPSP. (13)		Undec. (6)
SUM	Empt.	PSP.-c (16)	Undec. (15)	Open (22)
	Univ.	Undec. (15)	Undec. (6)	Undec. (6)
SUM$^+$	Empt.	PSP.-c (16)		EXPSP. (19)
	Univ.	EXPSP. (16)		Undec. (6)

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

		INF / LIMINF	SUP / LIMSUP	LIMAVG
MIN, MAX SUM^B	Empt.	PTIME		
	Univ.	PSPACE-c		
SUM	Empt.	PTIME	Undec.	Open (22)
	Univ.	Undec.	PSPACE-c	
SUM$^+$	Empt.	PTIME		
	Univ.	PSPACE-c		

Table 3: Decidability and complexity of the emptiness and universality problems for functional $(f;g)$-automata whose slave automata have size bounded by a constant. The alignment is as in Table 1.

		INF / LIMINF	SUP / LIMSUP	LIMAVG
MIN, MAX SUM^B	Empt.	PTIME		
	Univ.	PSPACE-c		Undec.
SUM	Empt.	PTIME	Undec.	Open (22)
	Univ.	Undec.	Undec.	Undec.
SUM$^+$	Empt.	PTIME		
	Univ.	PSPACE-c		Undec.

Table 4: Decidability and complexity of the emptiness and universality problems for non-deterministic $(f;g)$-automata whose slave automata have size bounded by a constant. The alignment is as in Table 1.

In fact, in case (2), 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 case (2) is undecidable. As automata in case (1) are equivalent to $\mathsf{sil}(f)$-automata for $f \in \{\textsc{Inf}, \textsc{LimInf}, \textsc{Sup}, \textsc{LimSup}\}$ (Lemma 10), which are essentially equivalent to f-automata, the inclusion problem is decidable [12].

Remark 23 (Parametric complexity)**.** *The complexity results summarized in Tables 1 and 2 are given w.r.t. the size of a nested automaton, i.e., the sum of the size of the master automaton and the total size of slave automata. However, if the total size of slave automata is bounded by a constant, then we show that the complexity of all emptiness problems decreases from* PSPACE *(resp.* EXPSPACE*) to* PTIME*, and all the universality problems become* PSPACE*-complete (Remarks 11 and 39). In other words, we show that the complexity of emptiness and universality in the size of the master automaton (with the total size of slave automata considered as constant) matches that of Boolean non-nested automata. (For every $f \in \mathsf{InfVal}$ the universality problem for functional f-automata is* PSPACE*-complete [19]). Interestingly, bounding the total size of slave automata does not change decidability status; all undecidability results still hold. The parametric complexity results are summarized in Table 3 and Table 4.*

5 Applications

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

5.1 Quantitative system properties

We have shown (Example 5) that basic properties such as average response time can be expressed conveniently as a nested weighted automaton. We also 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.

Quantitative monitor automata. In verification of Boolean properties, the formalism with *monitor automata* is a very convenient way to express system properties [26]. The specification for a system can be decomposed into subproperties, each monitor automaton tracks a subproperty, and the logical value of the specification is inferred from the results of the monitor automata. To be more specific, given an LTL specification, the logical value of every subformula is tracked by a monitor automaton. A monitor automaton is a transducer that at each position of the word outputs whether the current suffix satisfies the given subformula. The monitor automata for complex formulae are constructed from monitor automata for their immediate subformulae. Finally, the answer whether a given word satisfies the specification is encoded as the first output of the monitor that corresponds to the whole LTL formula. Our nested weighted automata framework can be seen as a natural extension of the formalism provided by monitor automata. Below we argue how nested weighted automata provide a convenient framework for specification, with added expressiveness, and is a first step towards quantitative run-time verification.

- *Ease of specification.* A specification formalism is a convenient framework if complex specifications can be easily decomposed. For Boolean properties, monitor automata were introduced for this purpose: in other words, for Boolean properties, though monitor are not more expressive than the standard automata, yet they are widely used as they provide a framework where specifications can be conveniently described. In our setting, in the spirit of 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. This (as in the case of monitor automata) allows decomposing quantitative properties into subproperties and thus eases the task of specification. For example as shown in Example 5 to compute average response time, for each request the master automaton invokes a slave automaton that computes the response time (a subproperty for every request) and then the master automaton with limit-average value function combines the subproperties to obtain the average response time.

  Example 24 (Average resource consumption)**.** *Consider a system with at most n concurrently running processes, in which processes can be started and terminated. The system has available resources $r_1, \ldots, r_k$. The quantitative property of* average resource consumption*, which asks what is the average number of different resources allocated by processes, can be expressed in a convenient way by a (deterministic) nested weighted automaton $\mathbb{A}$ defined as follows. The master automaton of $\mathbb{A}$ starts a separate slave automaton*

$\mathfrak{B}$ *when a new process is started. The slave automaton $\mathfrak{B}$ runs until the process terminates and counts how many different resources $r_1, \ldots, r_k$ the given process allocates. The counting can be implemented by a* MAX*-automaton with weights $\{0, 1, \ldots, k\}$. Then, the master automaton computes the limit average of resource consumption computed by slave automata. Since every* (LIMAVG; MAX)*-automaton is equivalent to some* sil(LIMAVG)*-automaton (Lemma 10), average resource consumption can also be expressed by a weighted automaton. However, construction of such a weighted automaton is cumbersome and it essentially follows the proof of Lemma 10.*

We use Example 24 to show that (deterministic) nested weighted automata can be exponentially more succinct than (non-deterministic) weighted automata. Let $\mathsf{ARC}(n)$ denote the average resource consumption property for n-processes.

Theorem 25. *There is a deterministic* (LIMAVG; MAX)*-automaton of the size $O(n)$ expressing* $\mathsf{ARC}(n)$, *while every non-deterministic* sil(LIMAVG)*-automaton expressing* $\mathsf{ARC}(n)$ *has* $2^{\Omega(n)}$ *states.*

Proof. Recall the the automaton $\mathbb{A}$ from Example 24 that expresses average resource consumption. Its size is linearly bounded in the number of processes n. It remains to show that every sil(LIMAVG)-automaton expressing $\mathsf{ARC}(n)$, average resource consumption for n processes, has $2^{\Omega(n)}$ states. To show that, we need to give a more precise description of the system in consideration and its modeling.

We assume for simplicity that there is only a single resource. Each process $i \in \{1, \ldots, n\}$ is associated with the following actions: *start* (s_i), *allocation* of the resource (a_i), and *termination* (t_i). Formally, average resource consumption in w is defined as the limit average over all positions p at which a process starts $w[p] = s_i$ (for some i) of the indicator (0/1) whether a_i occurs in w between position p and the first occurrence of t_i past p.

We show that unless a sil(LIMAVG)-automaton has at least $2^{0.5n}$ states, it cannot compute average resource consumption. Assume towards contradiction that a sil(LIMAVG)-automaton $\mathcal{A}$ has less than $2^{0.5n}$ states and computes average resource consumption. For every $A \subseteq \{a_1, \ldots, a_n\}$, we define a word $u_A \in A^*$ as a periodic listing of all letters from A $|\mathcal{A}|$ times, i.e., $(b_1 \ldots b_s)^{|\mathcal{A}|}$, where $\{b_1, \ldots, b_s\} = A$. Consider execution traces $w_A = (su_A t)^\omega$, where $s = s_1 \ldots s_n$, $t = t_1 \ldots t_n$. Given a word w_A, let π_A be a run of $\mathcal{A}$ on w_A of the minimal value. Due to periodicity of w_u, such a run exists. We show the following claim:

(*) There exist cycles c_A, c_B in the automaton $\mathcal{A}$ such that (1) c_A, c_B are labeled with words over different alphabets A, B, with $|A| = |B| = 0.5n$, (2) c_A, c_B share a state q that occurs in both runs π_A, π_B with positive density, and (3) each of c_A, c_B is either silent or its average weight is $0.5n$.

We shall prove (*) later; first we show that (*) implies that $\mathcal{A}$ does not express average resource consumption. Indeed, consider a pair c_A, c_B from (*) and a_i from $B \setminus A$. We insert into the run π_A the cycle c_B at all positions where q occurs. Let π'_A be the resulting run, and let w'_A be the word that corresponds to π'_A. The resulting run π'_A has the same value as π_A, $\frac{1}{2}$, but average resource consumption in w'_A is higher. Indeed, in all blocks sut, for some u, in which u is different from u_A, the number of different letters in u is at least $|A| + 1$. In the remaining blocks, the number of different letters is $|A|$. Since blocks sut with $u \neq u_A$ occur with positive density, average resource consumption in w'_A is strictly higher than $\frac{1}{2}$. But, the value of w'_A does not exceed $\frac{1}{2}$ as π'_A is an accepting run on w'_A of the value $\frac{1}{2}$. It follows that $\mathcal{A}$ does not express the average resource consumption property.

Now, we prove (*). Consider a word w_A and an occurrence of u_A at position p in w_A. Since $|u_A| > |\mathbb{A}| + 2 \cdot |A|$, there is a state q that occurs twice in the run π_A between positions p and $p + |u_A|$ (the part corresponding to the considered occurrence of u_A) and the distance between occurrences of q between $|A|$ and $|u_A| - |A|$. These occurrences of q indicate a cycle, $c_{A,p}$, in $\mathbb{A}$, which is labeled with all letters from A. Indeed, among any $|A|$ consecutive letters in u_A each letter from A occurs. Now, we select c_A from cycles $c_{A,i}$, where i varies, that occurs with positive density in π_A. Let q_A be the state that occurs in c_A.

As there are more than $2^{0.5n}$ subsets of $\{a_1, \ldots, a_n\}$ of cardinality $0.5n$, there is a state that occurs in two cycles c_A, c_B with $A \neq B$. It remains to show that c_A, c_B are either silent or their average weight is $\frac{1}{2}$. If the average weight of c_A is greater than $\frac{1}{2}$, we can decrease the value of π_A by removing all occurrences of c_A. Recall that the length of c_A is at most $|u_A| - |A|$, therefore if we remove all parts of w_A that correspond to c_A, each process in the resulting word has resource consumption $0.5n$, hence average resource consumption is still $\frac{1}{2}$. But, the value of the corresponding run is lower than $\frac{1}{2}$, a contradiction. Conversely, if the value is

lower than $\frac{1}{2}$, we can pump that cycle to obtain a run of the value smaller than $\frac{1}{2}$ on a word whose resource consumption for each process is $0.5n$. Thus, c_A is either silent or its average weight value is $\frac{1}{2}$. $\qquad\qquad\square$

- *Expressiveness.* More importantly, as mentioned above, for Boolean properties, monitor automata only add convenience but not expressiveness, whereas we show that for quantitative properties, nested weighted automata are strictly more expressive than non-nested weighted automata. Moreover, we show that the added expressiveness of nested weighted automata comes with the ability to express natural quantitative properties (like average response time) that could not be expressed as non-nested weighted automata.

- *Quantitative run-time verification.* Finally, monitor automata are specially useful for safety properties, and widely used in run-time verification [20]. 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 [21] 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 (resp. weighted automata as in [21]). 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 [21]. We now show how nested weighted automata can define interesting similarity measures d_M.

Example 26 (Bounded delays). *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. Also 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

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

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 similarity measure d_M is defined as a (SUP; SUM$^+$)*-automaton $\mathbb{A}_D$ that computes the maximum delay in the following way. When a packet is sent, the master automaton starts a slave SUM$^+$-automaton that counts the number of transition until the packet is delivered. If 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.*

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 (resp. 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 27 (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* (SUP; 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.*

Remark 28 (Decidability of examples)**.** *Note that for all examples presented in the paper, they belong to the class of nested weighted automata for which we establish decidability of the emptiness problem.*

Remark 29 (Robustness of nested weighted automata)**.** *The model of nested weighted automata is robust with respect to several changes, e.g., (i) instead of labeling function on transitions we can have labeling function on states; or (ii) instead of invoking one slave automaton in every transition a constant number of slave automata can be invoked. These changes do not change the expressive power, nor the decidability and the complexity results for nested weighted automata.*

6 Emptiness of (LIMAVG; SUM$^+$)-automata is in EXPSPACE

In this section we prove that the emptiness problem for (LIMAVG; SUM$^+$)-automata is in EXPSPACE ((1) from Theorem 19), as the proof itself is interesting and requires new and non-standard techniques. We first present an overview of the proof.

Overview of the proof. The key argument will be to *simulate* a given (LIMAVG; SUM$^+$)-automaton $\mathbb{A}$ by a sil(LIMAVG)-automaton, however, the main conceptual difficulty is that (LIMAVG; SUM$^+$)-automata are strictly more expressive than sil(LIMAVG)-automata. We circumvent this problem (which is non-standard for weighted automata) in the following way:

1. *Step 1.* We establish a property $\mathfrak{C}$ on runs of a (LIMAVG; SUM$^+$)-automaton $\mathbb{A}$ such that (a) the infimum over values of runs satisfying $\mathfrak{C}$ is the same as the infimum over values of all runs, and (b) there is a sil(LIMAVG)-automaton that simulates $\mathbb{A}$ on runs satisfying $\mathfrak{C}$.

2. *Step 2.* We give the construction of a sil(LIMAVG)-automaton $\mathcal{A}$ specified in the condition (b) from Step 1.

   Although, $\mathcal{A}$ simulates $\mathbb{A}$, weighted automata and nested weighted automata accumulate weights in a different way; a run of $\mathbb{A}$ that satisfies $\mathfrak{C}$ and the corresponding run of $\mathcal{A}$ can have different values.

3. *Step 3.* We show that the infima over values of all runs of $\mathbb{A}$ and $\mathcal{A}$ are equal.

Proof of Step 1. We first introduce the notion of bounded multiplicity.

Configuration and multiplicities. In nested weighted automata, starting a slave automaton can be seen as a universal transition in the sense of alternating automata. We adapt the power-set construction, which is used to convert alternating automata to non-deterministic automata, to the nested weighted automata case. Given a nested weighted automaton $\mathbb{A}$, we define *configurations* and *multiplicities* of $\mathbb{A}$ as follows. Let Q_{slv} be the disjoint union of the sets of states of all slave automata of $\mathbb{A}$. For a run of $\mathbb{A}$, we say that (q_m, A) is the *configuration* at position p if q_m is the state of the master automaton at position p and $A \subseteq Q_{\text{slv}}$ is the set of states of slave automata at position p. We denote by $\text{conf}(\mathbb{A})$ the number of configurations of $\mathbb{A}$. We define the *multiplicity* mult at position p as the function mult $: Q_{\text{slv}} \mapsto \mathbb{N}$, such that $\text{mult}(q)$ specifies the number of slave automata in the state q at position p. The configuration together with the multiplicity give a complete description of the state of $\mathbb{A}$ at position p.

Optimal runs. The general idea to solve the emptiness problem for (LIMAVG; SUM$^+$)-automata is to simulate a given (LIMAVG; SUM$^+$)-automaton by a sil(LIMAVG)-automaton that keeps track of configurations and multiplicities. Unfortunately, unbounded multiplicities cannot be encoded in a finite set of states of a sil(LIMAVG)-automaton. But, the emptiness problem can be solved by inspecting only selected runs. More precisely, given a

(LIMAVG; SUM$^+$)-automaton $\mathbb{A}$, we say that runs satisfying a condition $\mathfrak{C}$ are *optimal* for $\mathbb{A}$ iff the infima over values of all runs of $\mathbb{A}$ and runs that satisfy $\mathfrak{C}$ are equal. We identify a condition such that the runs satisfying it are optimal and can be simulated by a sil(LIMAVG)-automaton. First, we observe that without loss of generality we can assume that nested weighted automata are deterministic. Basically, non-deterministic choices can be encoded in the input alphabet.

Lemma 30. *Given a* (LIMAVG; SUM$^+$)*-automaton $\mathbb{A}$ over Σ, one can compute in polynomial space a deterministic* (LIMAVG; SUM$^+$)*-automaton $\mathbb{A}'$ over an alphabet $\Sigma \times \Gamma$ such that* $\inf_{w \in \Sigma^+} \mathcal{L}_{\mathcal{A}}(w) = \inf_{w' \in (\Sigma \times \Gamma)^+} \mathcal{L}_{\mathcal{A}'}(w')$. *Moreover,* $\mathsf{conf}(\mathbb{A}) = \mathsf{conf}(\mathbb{A}')$.

Proof. The proof consists of two steps. We show that (i) for every run $(\Pi, \pi_1, \pi_2, \dots)$ of $\mathbb{A}$ there exists a *simple* run of $\mathbb{A}$ of the value not exceeding the value of $(\Pi, \pi_1, \pi_2, \dots)$. Next, we show that (ii) there exists a deterministic (LIMAVG; SUM$^+$)-automaton $\mathbb{A}'$ over an extended alphabet such that the sets of accepting simple runs of $\mathbb{A}$ and accepting runs of $\mathbb{A}'$ coincide and each run has the same value in both automata. Then (i) and (ii) imply the lemma statement.

(i): A run of a nested weighted automaton is *simple* if at every position in the run slave automata that are in the same state take the same transition. Now, consider a run $(\Pi, \pi_1, \pi_2, \dots)$ of $\mathbb{A}$. Suppose that 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 in 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 a simple run of the value not exceeding the value of $(\Pi, \pi_1, \pi_2, \dots)$.

(ii): Without loss of generality, we can assume that for every slave automaton in $\mathbb{A}$ final states have no outgoing transitions. Let Q_{all} be the disjoint union of the sets of states of the master automaton and all slave automata of $\mathbb{A}$. We define Γ as the set of all partial functions $h : Q_{\mathrm{all}} \mapsto Q_{\mathrm{all}}$. We define a (LIMAVG; SUM$^+$)-automaton $\mathbb{A}'$ over the alphabet $\Sigma \times \Gamma$ by modifying only the transition relations and labeling functions of the master automaton and slave automata of $\mathbb{A}$; the sets of states and accepting states are the same as in the original automata. The transition relation and the labeling function of the master automaton $\mathcal{A}'_{mas}$ of $\mathbb{A}'$ is defined as follows: for all states q, q', $(q, \langle a, h \rangle, q')$ iff $h(q) = q'$ and $\mathcal{A}_{mas}$ has the transition (q, a, q'). The label of the transition $(q, \langle a, h \rangle, q')$ is the same as the label of the transition (q, a, q') in $\mathcal{A}_{mas}$. Similarly, for each slave automaton $\mathfrak{B}_i$ in $\mathbb{A}$, the transition relation of the corresponding slave automaton $\mathfrak{B}'_i$ in $\mathbb{A}'$ is defined as follows: for all states q, q' of $\mathfrak{B}'_i$, $(q, \langle a, h \rangle, q')$ iff $h(q) = q'$ and $\mathfrak{B}_i$ has the transition (q, a, q'). The label of the transition $(q, \langle a, h \rangle, q')$ is the same as the label of the transition (q, a, q') in $\mathfrak{B}_i$.

First, we see that $\mathsf{conf}(\mathbb{A}) = \mathsf{conf}(\mathbb{A}')$. Second, observe that the master automaton $\mathcal{A}'_{mas}$ and all slave automata $\mathfrak{B}'_i$ are deterministic. Moreover, since we assumed that for every slave automaton in $\mathbb{A}$ final states have no outgoing transitions, slave automata $\mathfrak{B}'_i$ recognize prefix free languages. Finally, it follows from the construction that (i) for every simple run $(\Pi, \pi_1, \pi_2, \dots)$ of $\mathbb{A}$ is also a run of $\mathbb{A}$ of the same value. One needs to encode non-deterministic transitions in functions $h \in \Gamma$. The value of each transition is the same by the construction. Conversely, (ii) a run $(\Pi, \pi_1, \pi_2, \dots)$ of $\mathbb{A}'$ is a simple run of $\mathbb{A}$ of the same value. Indeed, the fact that transitions are directed by functions $h \in \Gamma$ implies that the run is simple. $\square$

We attempt to simulate (LIMAVG; SUM$^+$)-automaton $\mathbb{A}$ by a sil(LIMAVG)-automaton. For that, we need to show that runs with bounded multiplicities or bounded values returned by slave automata are optimal for $\mathbb{A}$; otherwise the state space of the simulating automaton would have to be unbounded. However, such a direct statement does not hold as we see in the following example.

Example 31. *Consider a* (LIMAVG; SUM$^+$)*-automaton $\mathbb{A}$ over $\{a, b\}$ such that on letter a (resp. b) the master automaton starts a slave automaton $\mathfrak{B}_a$ (resp. $\mathfrak{B}_b$). The automaton $\mathfrak{B}_a$ accepts words $a^* b$, and for a word $a^k b$ assigns value $k \mod 2$. The automaton $\mathfrak{B}_b$ accepts words $ba^* b$, and for a word $ba^k b$ assigns value $k + 2$. Observe that the infimum over all runs of $\mathbb{A}$ is $\frac{3}{2}$.*

At the end of a block of k letters a, the number of slave automata running concurrently is $k + 1$, one automaton $\mathfrak{B}_b$ and k automata $\mathfrak{B}_a$, and the value returned by $\mathfrak{B}_b$ is $k + 2$. It follows that if the multiplicity of a run is bounded by $k + 1$ or the maximal returned values are bounded by $k + 2$, lengths of all block of a's are bounded by k. However, if the length of block's of letter a are bounded by k, the value of such a run is at least $\frac{3 \cdot k + 4}{2 \cdot (k+1)}$. Thus, runs of bounded multiplicity or bounded returned value are not optimal for $\mathbb{A}$.

Example 31 shows that we cannot bound the number of slave automata running concurrently or the values returned by slave automata. However, we can combine these two conditions, i.e., we show that while computing the infimum over values of runs of an (LIMAVG; SUM$^+$)-automaton, there is a constant $\mathbf{N}$ such that we can discard runs in which more than $\mathbf{N}$ slave automata accumulate value above $\mathbf{N}$. Then, slave automata that return bounded values are essentially bounded sum automata, and can be eliminated, and only bounded number of slave automata returning unbounded values remain. E.g. the automaton $\mathbb{A}$ from Example 31 is equivalent to a (LIMAVG; SUM$^+$)-automaton $\mathbb{A}^{\#}$ that, instead of starting a slave automaton $\mathfrak{B}_a$, guesses the parity of the following block of letters a and, based on that guess, starts $\mathfrak{B}_{a0}$ or $\mathfrak{B}_{a1}$, which terminates after a single transition and returns 0 ($\mathfrak{B}_{a0}$) or 1 ($\mathfrak{B}_{a1}$). The master automaton verifies the correctness of the guessed parity.

Synchronized silent transitions. The bounded multiplicity property enables simulation of $\mathbb{A}$ by sil(LIMAVG)-automaton, but it does not guarantee that the corresponding runs have the same value. We impose the following condition on optimal runs of $\mathbb{A}$. We say that a run of $\mathbb{A}$ has *synchronized silent transitions* if at every position where the master automaton takes a silent transition, each slave automaton takes a transition of weight 0. On runs of $\mathbb{A}$ with synchronized silent transitions, the sil(LIMAVG)-automaton can take a silent transition whenever the master automaton of $\mathbb{A}$ takes a silent transition as no weighted transition is lost. To achieve synchronization of silent transitions, we modify slave automata so that during silent transitions of the master automaton, slave automata accumulate their values in their states, while taking transitions of weight 0, and flush the accumulated value λ by taking transition of weight λ once the master automaton takes a non-silent transition. We prove that runs with sequences of silent transitions bounded by conf($\mathbb{A}$) are optimal for $\mathbb{A}$, therefore slave automata have to accumulate only bounded weights.

We combine the ideas for bounding the multiplicity and synchronization of silent transitions in the following lemma.

Lemma 32. *Let $\mathbb{A}$ be a deterministic (LIMAVG; SUM$^+$)-automaton. There is a constant $\mathbf{c}$ quadratically bounded in conf($\mathbb{A}$) and a deterministic (LIMAVG; SUM$^+$)-automaton $\mathbb{A}_0$ equivalent to $\mathbb{A}$ such that runs that have (1) multiplicities bounded by $\mathbf{c}$, and (2) synchronized silent transitions, are optimal for $\mathbb{A}_0$. The size of $\mathbb{A}_0$ is exponentially bounded in $|\mathbb{A}|$.*

Before we prove Lemma 32, we show its vital components:

Lemma 33. *Let $\mathbb{A}$ be a deterministic (LIMAVG; SUM$^+$)-automaton that has an accepting run. Runs such that among every consecutive conf($\mathbb{A}$) steps, the master automaton of $\mathbb{A}$ takes a non-silent transition are optimal for $\mathbb{A}$.*

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

Observe that there are positions $i < i', j' < j$ with the same configuration (defined in Section 6). 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 serious 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. $\square$

Lemma 34. *Let $\mathbb{A}$ be a deterministic (LIMAVG; SUM$^+$)-automaton that recognizes a non-empty language. Let $\mathbf{N} = (|Q_s| + 2) \cdot \text{conf}(\mathbb{A})$. The runs that eventually (for every position s greater than some position s_0) satisfy the following condition (*) are are optimal for $\mathbb{A}$: (*) among slave automata active at position s, at most $2 \cdot \mathbf{N}$ will accumulate value greater than $4 \cdot \mathbf{N}$.*

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

Consider any 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 greater 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 greater 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 $\text{mult}_{uw[1,-|u|]} \upharpoonright_{\mathbf{N}} = \text{mult}_u \upharpoonright_{\mathbf{N}}$. 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 overlap with

any position from $\{j_0, \ldots, j_n\}$. 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 $\mathsf{mult}_{uv}(q) \leq \mathsf{mult}_u(q) \upharpoonright_{\mathbf{N}}$. 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 $\mathsf{mult}_{uw[1,j]} \upharpoonright_{\mathbf{N}} = \mathsf{mult}_u \upharpoonright_{\mathbf{N}}$. 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, does not increase the value 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 of all inactive slave automata plus the value of active slave automata accumulated up to the position $|u|$,

- $(2)'$ 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, for every run of $\mathbb{A}$ of value λ, there is s_0 and an a run of the value not exceeding λ such that at each position $s > s_0$ at most $2 \cdot \mathbf{N}$ will accumulate value greater than $4 \cdot \mathbf{N}$. $\square$

Proof of Lemma 32. We transform the automaton $\mathbb{A}$ to an equivalent deterministic ($\textsc{LimAvg}; \textsc{Sum}^+$)-automaton $\mathbb{A}_0$ for which runs that at each position (1) at most $\mathbf{c}$ slave automata run, and (2) if the master automaton takes a silent transition, each slave automaton takes a silent transition, are optimal. Due to Lemma 34 runs for which eventually at most $2 \cdot \mathbf{N}$ slave automata accumulate value greater than $4 \cdot \mathbf{N}$ are optimal for $\mathbb{A}$. Moreover, by Lemma 33 runs in which at least one in every $\mathsf{conf}(\mathbb{A})$ transitions is non-silent are optimal for $\mathbb{A}$.

We 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 past which at most $2 \cdot \mathbf{N}$ slave automata accumulate value greater than $4 \cdot \mathbf{N}$ are optimal for $\mathbb{A}_0$. Prior to that marking, a modified automaton starts only dummy slave automata that immediately terminate. Past that marking $\mathbb{A}_0$ 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 run concurrently. The automaton $\mathbb{A}_0$ is equivalent to $\mathbb{A}$, as the return values of slave automata past the position s_0 are the same and the $\textsc{LimAvg}$ value function does not depend on finite prefixes. Therefore, runs satisfying conditions (1) and (2) are optimal for $\mathbb{A}_0$. $\square$

The size of the automaton $\mathbb{A}_0$ from Lemma 32 is polynomial in the size of the master automaton of $\mathbb{A}$.

Proof of Step 2. We now prove Step 2 which basically involves the classic power-set construction.

Construction of $\mathcal{A}$. We adapt the classic power-set construction to construct a $\mathsf{sil}(\textsc{LimAvg})$-automaton $\mathcal{A}$ that simulates runs of a given $(\textsc{LimAvg}; \textsc{Sum}^+)$-automaton $\mathbb{A}$ with bounded multiplicities and synchronized silent transitions. Let $\mathbb{A}$ be a deterministic $(\textsc{LimAvg}; \textsc{Sum}^+)$-automaton. Without loss of generality (Lemma 32), we assume that runs with (1) multiplicities bounded by $\mathbf{c}$ and (2) synchronized silent transitions are optimal for $\mathbb{A}$. The automaton $\mathcal{A}$, which simulates runs satisfying (1) and (2), keeps track of the current configuration and the multiplicity of $\mathbb{A}$. That is, the set of states of $\mathcal{A}$ is $Q_m \times \mathbf{c}^{Q_{\mathsf{slv}}}$, where Q_m, Q_{slv} are respectively the set of states of the master automaton and the union of the sets of states of all slave automata of $\mathbb{A}$. The component $\mathbf{c}^{Q_{\mathsf{slv}}}$ encodes a part of configuration and multiplicity. For all states $(q_1, h_1), (q_2, h_2)$ of $\mathcal{A}$ and every letter a, $\mathcal{A}$ has a transition $\langle (q_1, h_1), a, (q_2, h_2) \rangle$ iff the master automaton has a transition (q_1, a, q_2) with the label i and the multiplicities h_2 follow from h_1 according to transitions of the slave automata and invocation of $\mathfrak{B}_i$. The weights of transitions of $\mathcal{A}$ are defined as follows. If the transition (q_1, a, q_2) of the master automaton of $\mathbb{A}$ is silent, the transition $\langle (q_1, h_1), a, (q_2, h_2) \rangle$ is silent. Otherwise, the weight of $\langle (q_1, h_1), a, (q_2, h_2) \rangle$ equals the sum of weights of the corresponding transitions of simulated slave automata multiplied by their multiplicities h_1. Recall that the simulated runs have silent transitions synchronized, therefore, the values accumulated by each slave automaton and the corresponding simulated automaton are equal.

Given a run of $\mathcal{A}$ one can construct a run of $\mathbb{A}$ with multiplicities bounded by $\mathbf{c}$, and vice versa. Hence, for a run of $\mathcal{A}$ (resp. $\mathbb{A}$) we refer to the corresponding run of $\mathbb{A}$ (resp. $\mathcal{A}$). We can solve the emptiness problem for $\mathcal{A}$ as $\mathsf{sil}(\textsc{LimAvg})$-automata behave similarly to $\textsc{LimAvg}$-automata:

Lemma 35. *(1) The emptiness problem for* $\mathsf{sil}(\textsc{LimAvg})$*-automata is in* $\textsc{NLogSpace}$*. (2) For every* $\mathsf{sil}(\textsc{LimAvg})$*-automaton $\mathcal{A}$ that recognizes a non-empty language, there is a run η of $\mathcal{A}$ such that (a) the value of η is minimal among values of all runs of $\mathcal{A}$, (b) at least one in every $|\mathcal{A}|$ transitions is non-silent, and (c) partial sums converge, i.e.,*

$$\limsup_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\eta))[i] = \liminf_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\eta))[i]$$

Proof. Let $\mathcal{A} = (\Sigma, Q, q_0, \delta, F, C)$ be a $\mathsf{sil}(\textsc{LimAvg})$-automaton. We show that if there is a run of $\mathcal{A}$ of the value not exceeding λ there is a lasso run such that the average weight in its cycle does not exceed λ, i.e. there is a run $\pi = \pi[0]\pi[1] \ldots \pi[n]$ of length $n \leq |\lambda|$ on a word $w = w[1] \ldots w[n]$ such that $\pi[0] = q_0$, for some $i < n$ we have $\pi[i] = \pi[n]$ (such a run π is called a lasso), $\pi[0], \ldots, \pi[n-1]$ are distinct and $\frac{1}{n-i}(C(\pi[i], w[i+1], \pi[i+1]) + \ldots + C(\pi[n-1], w[n], \pi[n])) \leq \lambda$. The existence of such a lasso can be decided in $\textsc{NLogSpace}$, which shows (1). As there are only finite number of values of lassos, there is λ_0 which is the smallest. It follows that there is no run of $\mathcal{A}$ of the value λ' smaller than λ_0; otherwise there would be a lasso of the value not exceeding λ'. Thus, the lasso of the value λ_0 has the minimal value among values of all runs, and the sequence of partial averages converge.

First, we define a $\textsc{LimAvg}$-automaton $\mathcal{A}_{\mathrm{fix}}$ 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 $\mathcal{A}_{\mathrm{fix}}$, and vice versa, every accepting run of $\mathcal{A}_{\mathrm{fix}}$ can be extended to a run of $\mathcal{A}$ by inserting silent transitions. The set of states of $\mathcal{A}_{\mathrm{fix}}$ is the same as $\mathcal{A}$ and the transition relation of $\mathcal{A}_{\mathrm{fix}}$ 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). It follows from the construction that $\mathcal{A}_{\mathrm{fix}}$ has the stipulated properties and their corresponding runs have the same value. Therefore, $\mathcal{A}_{\mathrm{fix}}$ has a lasso l_0 such that the average weight in its cycle does not exceed λ. The lasso l_0 can be extended to a lasso l_1 in $\mathcal{A}$, which can have states that occur multiple times. We can remove duplicate states in the following way, if the average weight between i and j with $l_1[i] = l_1[j]$ is above λ, we remove all the states between $i + 1$ and j. Otherwise, if the average weight does not exceed λ, we can remove all the states following $\pi[j]$. Hence, we have shown that if $\mathcal{A}$ has a run of the value λ it has a lasso such that the average weight in its cycle does not exceed λ. $\square$

Proof of Step 3. We now prove Step 3, i.e., we show that the emptiness problems for $\mathbb{A}$ and $\mathcal{A}$ coincide. The main problem is that, even though a run of $\mathbb{A}$ and the corresponding run of $\mathcal{A}$ represent the same sequences of weights, they can have different values. Still, we show that infima over values of all runs of $\mathbb{A}$ and $\mathcal{A}$ are equal.

Accumulation of weights. The automata $\mathbb{A}$ and $\mathcal{A}$ compute their values differently. In $\mathbb{A}$ a slave automaton started at a position k computes its value and returns it as a weight of the transition at position k, whereas in $\mathcal{A}$, simulated

slave automata run concurrently and add their weights to the partial sum at each step. To visualize this, consider a matrix such that the value at (i, j) is the weight of the transition of j-th slave automaton at position i in the input word. 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.
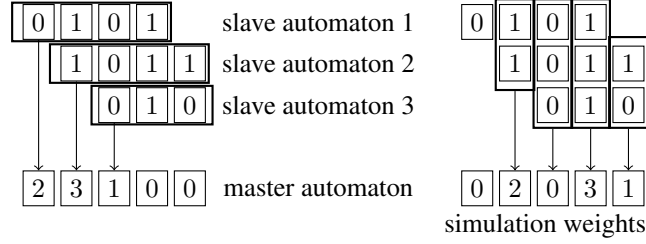

Figure 2: The matrix depicting the difference in aggregation of weights by $\mathbb{A}$ and $\mathcal{A}$.

In consequence, a run of $\mathbb{A}$ and the corresponding run of $\mathcal{A}$ can have different values.

Example 36. *Consider the automaton $\mathbb{A}^{\#}$ that has been discussed above as a modification of an automaton from Example 31 and a word $w = bab \ldots ba2^k-1 b \ldots$. At position $2^k + 1$, the partial sum $\sum_{i=1}^{k}(C(\pi_i))$ is equal to the sum of (1) the values of blocks $bab, \ldots, ba2^{k-1}-1b$ and (2) the value of $\mathfrak{B}_b$ started at the beginning of $ba2^k-1b$ equal $2^k - 1$. The value of a block $ba2^i-1b$ equals $2^i - 1$ (accumulated by $\mathfrak{B}_b$) plus $2^{i-1} - 1$ (accumulated by $\mathfrak{B}_{a1}$). Thus, the value (1) equals $2^k + 2^{k-1} - 2 \cdot k$, i.e., $2^k + 2^{k-1} - O(k)$. Therefore, the partial average at position $2^k + 1$ is $\frac{5}{2} - O(\frac{k}{2^k})$. Thus, the value of w assigned by $\mathbb{A}^{\#}$ is $\frac{5}{2}$. In contrast, the automaton $\mathcal{A}^{\#}$ simulating $\mathbb{A}^{\#}$ simply takes a transition of weight 0 on letter b, 1 on even occurrences of letter a and 2 on odd occurrences of letter a. The value of w assigned by $\mathcal{A}^{\#}$ is $\frac{3}{2}$.*

Nevertheless, we show that the emptiness problems for $\mathcal{A}$ and $\mathbb{A}$ coincide. In the following lemma we will use a notion of *reset words* used to terminate long runs of slave automata.

Reset words. Given a word w, a finite word u is a *reset word* for $\mathbb{A}$ at position i if in $w[1, i]uw[i + 1, \infty]$ we have (1) the configuration of $\mathbb{A}$ at positions i and $i + |u|$ is the same (recall that $\mathbb{A}$ is deterministic), and (2) all slave automata active at position i terminate before position $i + |u|$. We say that the word $w[1, i]uw[i + 1, \infty]$ is the result of injecting a reset word at position i. Observe that (1) implies that $\mathbb{A}$ accepts w iff it accepts $w[1, i]uw[i + 1, \infty]$. As in an accepting run of $\mathbb{A}$ past some finite position all configurations occur infinitely often and all slave automata terminate after finite number of steps, therefore at almost all positions p, there exists a reset word that can be injected at p. Basically, that word occurs already at position p. In addition, by simple (un)pumping argument we can show that for almost all positions there exist reset words with length bounded by $|Q_{\text{slv}}| \cdot \text{conf}(\mathbb{A})$.

Lemma 37. *The emptiness problems for $\mathbb{A}$ and $\mathcal{A}$ coincide.*

Proof. 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 $\mathcal{A}$ at every position k we have $\sum_{i=1}^{k}(C(\eta))[i] \leq \sum_{i=1}^{k}(C(\pi_i))$. Therefore, the infimum over runs of $\mathcal{A}$ does not exceed the infimum over runs of $\mathbb{A}$.

Conversely, due to Lemma 35, $\mathcal{A}$ has a run η of the minimal value and whose sequence of partial averages converges. First, we show that there is a run η' of the same value as η such that in the corresponding run of $\mathbb{A}$, for every k greater than some constant, each slave automaton started before position k, terminates before position $k + \log(k)$. Then, we show that η' and the run of $\mathbb{A}$ corresponding to η' have the same value.

Consider a sequence $\{a_i\}_{i \geq 0}$ where a_0 is the least position past which every configuration occurs infinitely often 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. The value of η' is the same as the value of η. Indeed, for almost all k we have

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

It follows from the fact that there are $|\{a_i : a_i < k\}| = o(k)$ reset words up to position k and the total increase of the partial sum due to 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 up to position k, the total increase of the partial sum due to injected reset words is $o(k)$. Due to Lemma 35, there are at least $\frac{k}{|\mathcal{A}|}$ non-silent transitions up to position k. Hence the values of η' do not exceed the value of η, which is minimal. Hence, the values of η, η' are equal.

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

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

However, each $(C(\eta'))[i]$ is bounded by a constant. Therefore, $\sum_{i=k+1}^{k+\log k} (C(\eta'))[i] = O(\log k)$. Again, as the number of non-silent transitions up to position k is $\Omega(k)$, we have $\lim_{k \to \infty} \frac{O(\log k)}{\Omega(k)} = 0$. Since $\mathbb{A}$ and $\mathcal{A}$ take non-silent transitions at the same positions, the limit averages of $(C(\eta'))[1], (C(\eta'))[2], \dots$ and $C(\pi_1), C(\pi_2), \dots$ are equal. Thus, the value of the infimum over runs of $\mathcal{A}$ does not exceed the value of the infimum over runs of $\mathbb{A}$. This implies that the emptiness problems for $\mathcal{A}$ and $\mathbb{A}$ coincide. $\square$

Finally, we prove the main statement.

Lemma 38. *The emptiness problem for* $(\text{LimAvg}; \text{Sum}^+)$-*automata is in* ExpSpace.

Proof. First, we transform a given $(\text{LimAvg}; \text{Sum}^+)$-automaton $\mathbb{A}$ to a deterministic $(\text{LimAvg}; \text{Sum}^+)$-automaton $\mathbb{A}^d$ (Lemma 30). The transformation does not change the cardinality of the set of configurations, i.e., $\mathcal{C}(\mathbb{A}) = \mathcal{C}(\mathbb{A}^d)$. Next, we transform $\mathbb{A}^d$ to an equivalent automaton $\mathbb{A}'$ for which runs with (1) multiplicities bounded by $\mathbf{c}$ and (2) synchronized silent transitions are optimal (Lemma 32). Finally, we define a $\text{sil}(\text{LimAvg})$-automaton $\mathcal{A}$ that simulates $\mathbb{A}'$ on runs that satisfy (1) and (2). Since the emptiness problems for $\mathbb{A}'$ and $\mathcal{A}$ coincide (Lemma 37), we solve the former.

Observe that the size of $\mathcal{A}$ is doubly exponential in the size of $\mathbb{A}$. Indeed, let Q'_{slv} be the disjoint union of the sets of states of the slave automata of $\mathbb{A}'$. The size of Q'_{slv} is exponential in the size of $\mathbb{A}$, therefore $|\mathbf{c}^{Q'_{\text{slv}}}|$ is doubly-exponential. However, the emptiness problem for $\mathcal{A}$ is decidable in NLogSpace (Lemma 35). Hence, the emptiness problem for $(\text{LimAvg}; \text{Sum}^+)$-automata is in ExpSpace. $\square$

Remark 39. *The transformations from Lemma 30 and Lemma 32 are polynomial in the size of the master automaton of a given nested automaton* $\mathbb{A}$. *Then, the number of configurations and, in consequence,* $\mathbf{c}$ *are polynomial in the size of the master automaton of* $\mathbb{A}$. *Finally, the set of states of the* $\text{sil}(\text{LimAvg})$-*automaton* $\mathcal{A}$ *constructed in Step 2 is* $Q_m \times \mathbf{c}^{Q_{slv}}$. *Thus, it is polynomial in the size of the master automaton of* $\mathbb{A}$ *as well as its weights. Therefore, the emptiness problem of* $(\text{LimAvg}; \text{Sum}^+)$-*automata is in* PTime *if the total size of slave automata is bounded by a constant.*

7 Related work

In this section we discuss various related work.

Weighted counterpart of Boolean nested automata. Weighted nested automata have been considered in [7] in the context of finite words, where the weights are given over semirings. It is further required that the semirings of all master and slave automata coincide, while in our case, the value functions may differ. Since the semirings of master and slave automata must coincide for [7], it can be interpreted as defining weighted counterpart of Boolean nested automata over finite words. Adding nesting structure to words and trees have been extensively studied for non-weighted automata in [2, 5] and also applied to software model checking [3]. The work of [7] defines a weighted counterpart of nesting of finite words, whereas we consider nesting of various 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 [7].

Weighted MSO Logics. Quantitative properties can be expressed in weighted logics, for example, *Weighted MSO Logics* [17] and weighted temporal logic [6]. For the basic decision problems for weighted logics over infinite words, the reduction is to weighted automata. For a given set of value functions that assigns values to infinite runs (such as FinVal and InfVal), weighted MSO logics are as expressive as weighted automata with the same class of

value functions [17]. It follows that with FinVal and InfVal as the value functions, weighted MSO logic cannot express the average response property. One can express average response time in weighted MSO logics by adding average response time itself as a primitive value function in the ω-valuation monoid. The decidability of weighted MSO logics with such a primitive can be established by a reduction to weighted automata that are able to express average response time, such as nested weighted automata. However, the reduction is non-elementary, as the basic decision problems for even non-weighted MSO logic have non-elementary complexity, whereas our complexity results range from PSPACE-complete to EXPSPACE.

Register automata. Another related model for specifying quantitative properties are *register automata* [4], which are parametrized by cost functions. The main differences between [4] and nested weighted automata are as follows: (i) register automata are over finite words, whereas we consider infinite words, and (ii) we consider nested control of automata, whereas register automaton are non-nested. As a result, both in terms of expressiveness and decidability results nested weighted automata are very different from register automata. For example, the emptiness of register automata with max and sum value functions is decidable, while we show emptiness to be undecidable for deterministic nested weighted automata with these value functions.

Other related models. Other possible quantitative models are visibly pushdown automata (VPA) with limit-average functions, or quantitative models consider in [24]. The framework of [24] neither captures the average response time property nor presents any decidability results. For VPA with limit-average functions it follows from [14] that even perfect-information VPA games (that correspond to simulation) with limit-average objectives are undecidable (the undecidability proof of [14] is for general pushdown games, but the proof itself also works for VPA games). Thus though there exist many other quantitative models, there exists no framework that can express the average response time property and have elementary-time complexity algorithms for the basic decision problems.

8 Conclusion and Future Work

Motivated by important system properties such as long-run average response time, we introduced the framework of nested weighted automata as a new, expressive, and convenient formalism for specifying quantitative properties of systems. We answered the basic decision questions for nested weighted automata. There are several directions for future work. First, we have an open conjecture (Conjecture 22) regarding the decidability of the emptiness of (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 automaton model, such as (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 distributions over runs and expected value of the runs, as in [10]).

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, S. Chaudhuri, and P. Madhusudan. Languages of nested trees. In *CAV*, pages 329–342, 2006.

[3] R. Alur, S. Chaudhuri, and P. Madhusudan. Software model checking using languages of nested trees. *ACM Trans. Program. Lang. Syst.*, 33(5):15, 2011.

[4] R. Alur, L. D'Antoni, J. V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22, 2013.

[5] R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.

[6] U. Boker, K. Chatterjee, T. A. Henzinger, and O. Kupferman. Temporal specifications with accumulative values. In *LICS*, pages 43–52, 2011.

[7] 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.

[8] K. Chatterjee, L. Doyen, H. Edelsbrunner, T. A. Henzinger, and P. Rannou. Mean-payoff automaton expressions. *CoRR*, abs/1006.1492, 2010.

[9] K. Chatterjee, L. Doyen, and T. A. Henzinger. Alternating weighted automata. In *FCT'09*, pages 3–13. Springer-Verlag, 2009.

[10] K. Chatterjee, L. Doyen, and T. A. Henzinger. Probabilistic weighted automata. In *CONCUR*, pages 244–258. LNCS 5710, Springer, 2009.

[11] K. Chatterjee, L. Doyen, and T. A. Henzinger. Expressiveness and closure properties for quantitative languages. *LMCS*, 6(3), 2010.

[12] K. Chatterjee, L. Doyen, and T. A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

[13] K. Chatterjee, T. A. Henzinger, and J. Otop. Nested weighted automata. In *LICS*, 2015.

[14] K. Chatterjee and Y. Velner. Mean-payoff pushdown games. In *LICS 2012*, pages 195–204, 2012.

[15] 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.

[16] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[17] M. Droste and I. Meinecke. Weighted automata and weighted mso logics for average and long-time behaviors. *Inf. Comput.*, 220:44–59, 2012.

[18] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer-Verlag New York, Inc., New York, USA, 1996.

[19] E. Filiot, R. Gentilini, and J.-F. Raskin. Quantitative languages defined by functional automata. In *CONCUR*, pages 132–146. LNCS 7454, Springer, 2012.

[20] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*, pages 342–356. LNCS 2280, Springer, 2002.

[21] T. A. Henzinger and J. Otop. From model checking to model measuring. In *CONCUR*, pages 273–287. LNCS 8052, Springer, 2013.

[22] D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977.

[23] H. Leung. Limitedness theorem on finite automata with distance functions: an algebraic proof. *Theor. Comput. Sci.*, 81(1):137–145, 1991.

[24] C. Mathissen. Weighted logics for nested words and algebraic formal power series. In *ICALP 2008*, pages 221–232, 2008.

[25] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. and Comb.*, 7(3):321–350, 2002.

[26] A. Pnueli and A. Zaks. On the merits of temporal testers. In *25 Years of Model Checking*, pages 172–195. LNCS 5000, Springer, 2008.