Bidirectional Nested Weighted Automata*

Krishnendu Chatterjee[1], Thomas A. Henzinger[2], and Jan Otop[3]

1 **IST Austria, Klosterneuburg, Austria**
krish.chat@ist.ac.at
2 **IST Austria, Klosterneuburg, Austria**
tah@ist.ac.at
3 **University of Wrocław, Wrocław, Poland**
jotop@cs.uni.wroc.pl

Abstract

Nested weighted automata (NWA) present a robust and convenient automata-theoretic formalism for quantitative specifications. Previous works have considered NWA that processed input words only in the forward direction. It is natural to allow the automata to process input words backwards as well, for example, to measure the maximal or average time between a response and the preceding request. We therefore introduce and study bidirectional NWA that can process input words in both directions. First, we show that bidirectional NWA can express interesting quantitative properties that are not expressible by forward-only NWA. Second, for the fundamental decision problems of emptiness and universality, we establish decidability and complexity results for the new framework which match the best-known results for the special case of forward-only NWA. Thus, for NWA, the increased expressiveness of bidirectionality is achieved at no additional computational complexity. This is in stark contrast to the unweighted case, where bidirectional finite automata are no more expressive but exponentially more succinct than their forward-only counterparts.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases weighted automata, nested weighted automata, complexity, bidirectional

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.5

1 Introduction

We study an extension of nested weighted automata (NWA) [13] that can process words in both directions. We show that this new and natural framework can express many interesting quantitative properties that the previous formalism could not. We establish decidability and complexity results of the basic decision problems for the new framework. We start with the motivation for quantitative properties, then describe NWA and our new framework, and finally the contributions.

Weighted automata. Automata-theoretic formalisms provide a natural way to express quantitative properties of systems. Weighted automata extend finite automata where every transition is assigned an integer number called weight. Thus a run of an automaton gives rise to a sequence of weights. A value function aggregates the sequence of weights into a single

* This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23,S11407-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award), ERC Start grant (279307: Graph Games), Vienna Science and Technology Fund (WWTF) through project ICT15-003 and by the National Science Centre (NCN), Poland under grant 2014/15/D/ST6/04543.

value. For non-deterministic weighted automata, the value of a word w is the infimum value of all runs over w. First, weighted automata were studied over finite words with weights from a semiring, and ring multiplication as value function [19], and later extended to infinite words with limit averaging or supremum as value function [12, 11, 10]. While weighted automata over semirings can express several quantitative properties [22], they cannot express long-run average properties that weighted automata with limit averaging can [12]. However, even weighted automata with limit averaging cannot express some basic quantitative properties (see [13]).

Nested weighted automata. A natural extension of weighted automata is to add nesting, which leads to *nested weighted automata (NWA)* [13]. A nested weighted automaton consists of a master automaton and a set of slave automata. The master automaton runs over input infinite words. At every transition the master can invoke a slave automaton that runs over a finite subword of the infinite word, starting at the position where the slave automaton is invoked. Each slave automaton terminates after a finite number of steps and returns a value to the master automaton. Each slave automaton is equipped with a value function for finite words, and the master automaton aggregates the returned values from slave automata using a value function for infinite words.

Advantages of NWA. We discuss the various advantages of NWA.

1. For Boolean finite automata, nested automata are equivalent to the non-nested counterpart, whereas NWA are strictly more expressive than non-nested weighted automata [13, Example 5]. NWA provide a specification framework where many basic quantitative properties can be expressed, which cannot be expressed by weighted automata [13].
2. NWA provide a natural and convenient way to express quantitative properties. Every slave automaton computes a subproperty, which is then combined using the master automaton. Thus NWA allow to decompose properties conveniently, and provide a natural framework to study quantitative run-time verification.
3. Finally, subclasses of NWA are equivalent in expressive power with automata with monitor counters [16], and thus they provide a robust framework to express quantitative properties.

Bidirectional NWA. Previous works considered slave automata that can only process input words in the forward direction (forward-only NWA). However, to specify quantitative properties, it is natural to allow slave automata to run backwards, for example, to measure the maximal or average time between a response and the preceding request. In this work we consider this natural extension of NWA, namely *bidirectional NWA*, where slave automata can process words in the forward as well as the backward direction.

Natural properties. First, we show that many natural properties can be expressed in the bidirectional NWA framework. We present two examples below (details in Section 3).

1. *Average energy level.* Consider a quantitative setting where each weight represents energy gain or consumption, and thus the sum of weights represents the energy level. To express the average energy level property, the master automaton has long-run average as the value function, and at every transition it invokes a slave automaton that walks backward with sum value function for the weights. Thus the average energy level property is naturally expressed by NWA with backward-walking slave automata, while this property is not expressible by NWA with forward-walking slave automata.

2. *Data-consistency property (DCP).* Consider the data-consistency property (DCP) where the input letters correspond to reads, writes, null instructions, and commits. For each read, the distance to the previous commit measures how fresh is the read with respect to the last commit, and this can be measured with a backward-walking slave automaton. For each write, the distance to the next commit measures how fresh is the write with respect to the following commit, and this can be measured with a forward-walking slave automaton. Thus the average freshness, called DCP, is expressed with bidirectional NWA. Moreover, the DCP can neither be expressed by NWA with only forward-walking slave automata nor by NWA with only backward-walking slave automata.

Our contributions. We propose bidirectional NWA as a specification framework for quantitative properties. First, we show that the classes of forward-only NWA and backward-only NWA have incomparable expressiveness, and bidirectional NWA strictly generalize both classes. Second, we establish complexity of the emptiness and universality problems for bidirectional NWA, where we consider the limit-average value function for the master automaton and for the slave automata we consider standard value functions for finite words (such as min, max, and variants of sum). The obtained complexity results coincide with the results for forward-only NWA, and range from NLogSpace-complete, PTime to PSpace-complete to ExpSpace. However the proofs for bidirectional NWA are much more involved than forward-only NWA. Thus bidirectional NWA have all the advantages of NWA but provide a more expressive framework for natural quantitative properties. Moreover, the added expressiveness of bidirectionality is achieved with no increase in the computational complexity of the decision problems (Table 1). We highlight two significant differences as compared to the unweighted case: (1) In the unweighted case bidirectionality does not change expressiveness, whereas we show for NWA it does; and (2) in the unweighted case for deterministic automata bidirectionality leads to exponential succinctness and increase in complexity of the decision problems, whereas for NWA bidirectionality does not change the computational complexity. Thus the combination of nesting and bidirectionality is very interesting in the weighted automata setting, which we study in this work.

Related works. Quantitative automata and logic have been extensively studied in recent years in many different contexts [19, 12, 4, 2]. The book [19] presents an excellent collection of results of weighted automata on finite words. Weighted automata on infinite words have been studied in [12, 11, 20]. Weighted automata over finite words extended with monitor counters have been considered (under the name of cost register automata) in [3, 21]. A version of nested weighted automata over finite words has been studied in [6], and nested weighted automata over infinite words has been studied in [13, 15, 14]. Several quantitative logics have also been studied, such as [5, 7, 1]. However, none of these works consider the rich and expressive formalism of quantitative properties expressible by NWA with slaves that walk both forward and backward, retaining decidability of the basic decision problems.

In the main paper, we present the key ideas and main intuitions of the proofs of selected results, and detailed proofs are presented in the full version [17]. Moreover, to ease of presentation we focus on bidirectional NWA where each slave automaton is either forward walking or backward walking. We can allow slave automata that change direction while running, i.e., allow two-way slave automata and obtain the same complexity results; we discuss nested weighted autmata with two-way slave automata in the full version [17].

2 Definitions

2.1 Words and automata

Words. We consider a finite *alphabet* of letters Σ. A *word* over Σ is a (finite or infinite) sequence of letters from Σ. We denote the i-th letter of a word w by $w[i]$, and for $i < j$ we define $w[i, j]$ as the word $w[i]w[i + 1] \dots w[j]$. The length of a finite word w is denoted by $|w|$; and the length of an infinite word w is $|w| = \infty$. For an infinite word w, word $w[i, \infty]$ is the suffix of w with first $i - 1$ letters removed. For a finite word w of length k, we define the reverse of w, denoted by w^R, as the word $w[k]w[k - 1] \dots w[1]$.

Labeled automata. For a set X, an *X-labeled automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, Q_0, \delta, F, C \rangle$, where (1) Σ is the alphabet, (2) Q is a finite set of states, (3) $Q_0 \subseteq Q$ is the set of initial states, (4) $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, (5) F is a set of accepting states, and (6) $C : \delta \mapsto X$ is a labeling function. A labeled automaton $\langle \Sigma, Q, \{q_0\}, \delta, F, C \rangle$ is *deterministic* if and only if δ is a function from $Q \times \Sigma$ into Q and Q_0 is a singleton.

Semantics of (labeled) automata. A *run* π of a (labeled) automaton $\mathcal{A}$ on a word w is a sequence of states of $\mathcal{A}$ of length $|w| + 1$ such that $\pi[0]$ belongs to the initial states of $\mathcal{A}$ and for every $0 \leq i \leq |w| - 1$ we have $(\pi[i], w[i + 1], \pi[i + 1])$ is a transition of $\mathcal{A}$. A run π on a finite word w is *accepting* if and only if the last state $\pi[|w|]$ of the run is an accepting state of $\mathcal{A}$. A run π on an infinite word w is *accepting* if and only if some accepting state of $\mathcal{A}$ occurs infinitely often in π. For an automaton $\mathcal{A}$ and a word w, we define $\mathsf{Acc}(w)$ as the set of accepting runs on w. Note that for deterministic automata, every word w has at most one accepting run ($|\mathsf{Acc}(w)| \leq 1$).

Weighted automata and their semantics. A *weighted automaton* is a $\mathbb{Z}$-labeled automaton, where $\mathbb{Z}$ is the set of integers. The labels are called *weights*. We define the semantics of weighted automata in two steps. First, we define the value of a run. Second, we define the value of a word based on the values of its runs. To define values of runs, we will consider *value functions* f that assign real numbers to sequences of integers. 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 denote by $(C(\pi))[i]$ the weight 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 the infimum of the empty set is infinite, i.e., the value of a word that has no accepting run is infinite. Every run π on the 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 $\perp$. Thus, the value of the empty word is either $\perp$, if the empty word is accepted by $\mathcal{A}$, or ∞ otherwise. To indicate a particular value function f that defines the semantics, we call a weighted automaton $\mathcal{A}$ with value function f an f-automaton.

Value functions. For finite runs we consider the following classical value functions: for runs of length $n + 1$ we have

- *Max and min:* $\textsc{Max}(\pi) = \max_{i=1}^{n}(C(\pi))[i]$ and $\textsc{Min}(\pi) = \min_{i=1}^{n}(C(\pi))[i]$.
- *Sum and absolute sum:* the sum function $\textsc{Sum}(\pi) = \sum_{i=1}^{n}(C(\pi))[i]$, the absolute sum $\textsc{Sum}^{+}(\pi) = \sum_{i=1}^{n} \mathsf{Abs}((C(\pi))[i])$, where $\mathsf{Abs}(x)$ is the absolute value of x.

- *Variants of bounded sum:* we consider a family of functions called the (variants of) bounded sum value function $\textsc{Sum}^{L,U}$. Each of these functions returns the sum if all the partial sums are in the interval $[L, U]$, otherwise there are many possibilities which lead to multiple variants. For example, we can require that for all prefixes π' of π we have $\textsc{Sum}(\pi') \in [L, U]$. We can impose a similar restriction on all suffixes, all infixes etc. Moreover, if partial sums are not contained in $[L, U]$, a bounded sum can return ∞, the first violated bound, etc.

For infinite runs we consider:

- *Limit average:* $\textsc{LimAvg}(\pi) = \liminf_{k\to\infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\pi))[i]$.

2.2 Nested weighted automata

Nested weighted automata (NWA) have been introduced in [13] and originally allowed slave automata to move only forward. The variant we define here allow two types of slave automata, forward walking and backward walking. The original definition of NWA from [13] is versatile and hence it can be seamlessly extended to the case with bidirectional (forward- and backward-walking) slave automata. We follow the description of [13].

Informal description. A *nested weighted automaton* consists of a labeled automaton over infinite words, called the *master automaton*, a value function f for infinite words, and a set of weighted automata over finite words, called *slave automata*. A nested weighted automaton can be viewed as follows: given a word, we consider the run of the master automaton on the word, but the weight of each transition is determined by dynamically running slave automata; and then the value of a run is obtained using the value function f. That is, the master automaton proceeds on an input word as an usual automaton, except that before taking 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 master automaton in the input word and works on some finite part of it. There are two types of slave automata: (a) forward walking, which move onward the input word (toward higher positions), and (b) backward walking, which move towards the beginning of the input word. Once a slave automaton finishes, it returns its value to the master automaton, which treats the returned value as the weight of the current transition that is being executed. The slave automaton might immediately accept and return value $\perp$, which corresponds to a *silent transition*, i.e., transition with no weight. If one of slave automata rejects, the nested weighted automaton rejects. We present two examples of properties expressible by NWA. Additional examples are presented in Section 3.

▶ **Example 1** (Average response time and its dual)**.** Consider infinite words over $\{r, g, \#\}$, where r represents requests, g represents grants, and $\#$ represents idle. A basic and interesting property is the average number of letters between a request and the corresponding grant, which represents the long-run *average response time (ART)* of the system. This property cannot be expressed by a non-nested automaton [13]. ART can be expressed by a deterministic nested weighted automaton, which basically implements the definition of ART. This automaton invokes at every request a forward-walking slave automaton with $\textsc{Sum}^{+}$ value function, which counts the number of events until the following grant. On the other events the NWA takes silent transitions. Finally, the master automaton applies $\textsc{LimAvg}$ value function to the values returned by slave automata. Figure 1 presents a run of the NWA computing ART.

We define the average workload property (AW), which measures the average number of pending requests. The average is computed over all positions in a word. Intuitively, if we

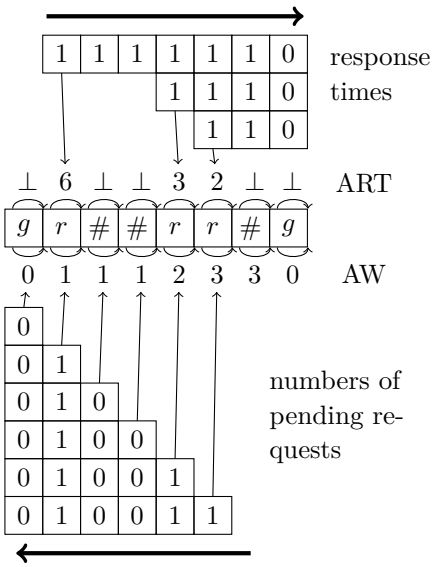

Figure 1 Runs of NWA computing ART (above) and AW (below). Each weight of a transition is dynamically computed as the sum of weights of slave automata. The thick arrows depict directions of slave automata.

pick a position in word w at random, the expected number of pending requests is the average workload of w. Formally, we define the workload at i in w, denoted $wl(w, i)$, as the number of letters r among $w[j, i]$, where j is the last position in $w[1, i]$ where g occurs or 1 if such a position does not exist. The average workload of w is the limit average over all positions i of $wl(w, i)$.

Observe that AW can be expressed by a deterministic (LimAvg; Sum$^+$)-automaton with backward-walking slave automata. Basically, the NWA invokes at every position a slave automaton, which counts the number of r letter from its current position to the first position containing letter g, where it terminates. Since slave automata run backwards, each of them computes the workload at the position of its invocation. Figure 1 presents a run of the NWA computing AW.

Now, we present a formal definition of NWA and their semantics.

Nested weighted automata. A *nested weighted automaton* (NWA) with bidirectional slave automata is a tuple $\langle \mathcal{A}_{mas}; f; \mathfrak{B}_{-m}, \ldots, \mathfrak{B}_0, \ldots, \mathfrak{B}_l \rangle$, with $m, l \in \mathbb{N}$ where (1) $\mathcal{A}_{mas}$, called the *master automaton*, is a $\{-m, \ldots, l\}$-labeled automaton over infinite words (the labels are the indexes of automata $\mathfrak{B}_{-m}, \ldots, \mathfrak{B}_l$), (2) f is a value function on infinite words, called the *master value function*, and (3) $\mathfrak{B}_{-m}, \ldots, \mathfrak{B}_l$ are weighted automata over finite words called *slave automata*. Intuitively, an NWA can be regarded as an f-automaton whose weights are dynamically computed at every step by the corresponding slave automaton. The automata $\mathfrak{B}_{-m}, \ldots, \mathfrak{B}_{-1}$ (resp., $\mathfrak{B}_1, \ldots, \mathfrak{B}_l$) are called *backward walking* (resp., *forward walking*) slave automata. We refer to NWA with both forward and backward walking slave automata as *bidirectional NWA*. The automaton $\mathfrak{B}_0$ immediately accepts and returns no weight; it is used to implement silent transitions, which have no weight. We define an $(f; g)$-*automaton* as an NWA where the master value function is f and all slave automata are g-automata.

Semantics: runs and values. A *run* of $\mathbb{A}$ on an infinite word w is an infinite sequence $(\Pi, \pi_1, \pi_2, \ldots)$ such that (1) Π is a run of $\mathcal{A}_{mas}$ on w; (2) for every $i > 0$ the label $j =$

$C(\Pi[i-1], w[i], \Pi[i])$ pointers at a slave automaton and (a) if $j < 0$, then π_i is a run of the automaton $\mathfrak{B}_j$ on some prefix of the reverse word $(w[1,i])^R$, and (b) if $j \geq 0$, then π_i is a run of the automaton $\mathfrak{B}_j$ on some finite prefix of $w[i,\infty]$. 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 takes its input sequence, removes $\perp$ symbols and applies f to the remaining sequence. 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 as f is a value function over infinite sequences, hence the sequence $v(\pi_1)v(\pi_2)\ldots$ with $\perp$ removed must be infinite.

Deterministic nested weighted automata. An NWA $\mathbb{A}$ is *deterministic* if (1) the master automaton and all slave automata are deterministic, and (2) in all slave automata, accepting states have no outgoing transitions. Intuitively, a slave automaton in an accepting state can choose (non-deteministically) to terminate or continue running; condition (2) removes this source of non-determinism.

Width of NWA. An NWA has *width* k if and only if in every run at every position at most k slave automata are active.

3 Examples

In this section we present several examples of quantitative properties that can be expressed with bidirectional NWA.

▶ **Example 2** (Average energy level)**.** We consider the average energy level property studied in [18, 8]. Consider $W \in \mathbb{N}$ and an alphabet Σ_W consisting of integers from interval $[-W, W]$. These letters correspond to the energy change, i.e., negative values represent energy consumption whereas positive values represent energy gain. For $w \in \Sigma_W$ we define the energy level at i as the sum $w[1] + \ldots + w[i]$. The average energy property (AE) is the limit average of the energy levels at every position. For example, the average energy level of $2(-1)3((-1)1)^\omega$ is 4.

AE can be expressed by a (LimAvg; Sum)-automaton $\mathbb{A}$ with backward-walking slave automata, but it is not expressible by (LimAvg; Sum)-automata with forward-walking slave automata. To express AE, a (LimAvg; Sum)-automaton $\mathbb{A}$ with backward-walking slave automata invokes at every position a slave automaton, which runs backward to the beginning of the word and sums up all the letters. In contrast, (LimAvg; Sum)-automata with forward-walking slave automata can use finite memory of the master automaton, but finite prefixes influence only finitely many values returned by slave slave automata and the limit-average value function neglects finite prefixes. Formally, we can show with a simple pumping argument that for every (LimAvg; Sum)-automaton with forward-walking slave automata, among words $w_i = 1^i 0^\omega$ there exists a pair of words with the same value. In contrast, all these words have different AE (AE of w_i is i).

AE property is often considered in conjunction with bounds on energy values. Typically, energy should not drop below some threshold, in particular, it should not be negative. In addition, the energy storage is limited, which motivates the upper bound on the stored energy,

where the excess energy is released. These two restrictions lead to the *interval constraint* on energy levels, i.e., we require the energy level at every position to belong to a given interval $[L, U]$, which results in a variant of the bounded sum $\textsc{Sum}^{L,U}$.

▶ **Example 3** (Data consistency)**.** Consider a database server, which processes instructions grouped into transactions. There are four instructions: read r, write w, void $\#$ and commit c. The commit instruction applies all writes, finishes the current transaction and starts a new one. The read instructions refer to writes applied before the previous commit.

In the presence of multiple clients connected to the database, there are two options to achieve consistency. One option is to use locks that can limit concurrency. A second approach is *optimistic concurrency* which proceeds without locks, and then rolls back in case there was a collision between transactions. In ordered to limit the number of roll backs, it is preferred that the read instructions occur shortly after commit, while write instructions are followed by the commit instruction as quickly as possible. Formally, we define (a) consistency (or freshness) of a read instruction as the number of steps to the first preceding commit instruction, and (b) consistency of a write instruction as the number of steps to the following commit instruction. The data consistency property (DCP) of w is the limit average of consistency of reads and writes in w.

DCP is expressed by the following deterministic $(\textsc{LimAvg}; \textsc{Sum}^{+})$-automaton $\mathbb{A}$ with bidirectional slave automata. On every read r (resp., w), the NWA $\mathbb{A}$ invokes a slave automaton which walks backward (resp., forward) and counts the number of steps to the first encountered c. On the remaining instructions $c, \#$, the NWA $\mathbb{A}$ invokes a dummy slave automaton which corresponds to a silent transition.

▶ **Example 4.** Consider the framework of Example 3. For every position with read r or write w we define a regret at position i as the minimal distance to the preceding or the following commit c. Intuitively, the regret corresponds to the number of instructions by which we have to prepone or postpone the commit to include the instruction at the current position. We consider the minimal regret property (MR) on words over $\{r, w, c, \#\}$ defined the limit average over positions with r and w of the regret at these positions. MR can be expressed by a non-deterministic $(\textsc{LimAvg}; \textsc{Sum}^{+})$-automaton with bidirectional slave automata, which basically implements the definition of MR (the non-deterministic guess is whether it is the preceding or the following commit). The NWA invokes at every r or w position one of the following two slave automata $\mathfrak{B}_B, \mathfrak{B}_F$. The automaton $\mathfrak{B}_B$ counts the number of steps to the preceding grant, while $\mathfrak{B}_F$ counts the number of steps to the following grant.

4 Decision questions

For NWA with bidirectional slave automata, we consider the quantitative counterparts of the fundamental problems of emptiness and universality. The (quantitative) emptiness and universality problems are defined in the same way for weighted automata and all variants of NWA; in the following definition $\mathcal{A}$ denotes either a weighted automaton or an NWA.

Emptiness and universality. Given an automaton $\mathcal{A}$ and a threshold λ, the *emptiness* (resp. *universality*) problem asks whether there exists a word w with $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$ (resp., for every word w we have $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$).

▶ Remark. The emptiness and universality problems have been studied for forward-only NWA in [13].

- For NWA the value functions considered for the master automaton are the infimum (or limit-infimum), the supremum (or limit-supremum), and the limit-average. For all the decidability results for the infimum (limit-infimum) and the supremum (limit-supremum) value functions the techniques are similar to unweighted automata [13], which can be easily adapted to the bidirectional framework. Hence in the sequel we only focus on bidirectional NWA with the limit-average value function for the master automaton.
- Moreover, we study only the emptiness problem for the following reasons. First, for the deterministic case the emptiness and the universality problems are similar and hence we focus on the emptiness problem. Second, in the non-deterministic case the universality problem is already undecidable for LimAvg-automata even with no nesting [9].

4.1 The minimum, maximum and bounded sum value functions

First, we show that for g being Min, Max, or a variant of the bounded sum value function $\text{Sum}^{L,U}$, the emptiness problem for (LimAvg; g)-automata with bidirectional slave automata is decidable in PSpace. To show that, we prove a stronger result, i.e., every (LimAvg; g)-automaton can be effectively transformed to a LimAvg-automaton of exponential size.

Key ideas. Weighted automata with value functions $\text{Min}, \text{Max}, \text{Sum}^{L,U}$ are close to (non-weighted) finite-state automata. In particular, these automata have finite range and for each value λ from the range, the set of words of value λ is regular. Thus, instead of invoking a slave automaton, the master automaton can non-deterministically pick value λ and verify that the value returned by this slave automaton is λ. For backward-walking slave automata the guessing can be avoided as the master automaton can simulate (the reverse of) runs of all backward-walking slave automata until the current position. Thus, we can eliminate slave automata from NWA, i.e., we transform such NWA to weighted automata. Formally, we show that for $g \in \{\text{Min}, \text{Max}, \text{Sum}^{L,U}\}$, every (LimAvg; g)-automaton with bidirectional slave automata can be transformed into an equivalent LimAvg-automaton of exponential size. The emptiness problem for non-deterministic LimAvg-automata is in NLogSpace (assuming weights given in unary) and hence we have the containment part in the following Theorem 5. The hardness part follows from PSpace-hardness of the emptiness problem for (LimAvg; g)-automata with forward-walking slave automata only [13].

▶ **Theorem 5.** *Let $g \in \{\text{Min}, \text{Max}, \text{Sum}^{L,U}\}$. The emptiness problem for non-deterministic* (LimAvg; g)*-automata with bidirectional slave automata is* PSpace*-complete.*

Note. The complexity in Theorem 5 does not depend on encoding of weights in slave automata, i.e., the problem is PSpace-hard even for a fixed set of weights, and it remains in PSpace for weights encoded in binary.

The average energy property from Example 2 with bounds on energy levels can be expressed with (LimAvg; $\text{Sum}^{L,U}$)-automata. The emptiness problem for these automata is decidable by Theorem 5.

▶ Remark (Parametrized complexity). If we assume that the size of slave automata in Theorem 5 is bounded by a constant, the complexity of the emptiness problem drops to NLogSpace-complete. NLogSpace-hardness follows from NLogSpace-hardness of the emptiness problem for LimAvg-automata, which can be considered as a special case of NWA.

The results of this section apply to general bidirectional NWA. In the following section we consider bidirectional NWA with the sum value function, where we consider additional

restrictions of finite width (Section 5) and bounded width (Section 6). We also justify in Remark 5.1 that the finite width restriction is natural.

5 Finite-width case

In this section we study NWA satisfying the *finite width* condition. First, we briefly discuss the finite-width condition and argue that it is a natural restriction. Next, we show that the emptiness problem for (finite-width) (LimAvg; Sum$^+$)-automata with bidirectional slave automata is decidable in ExpSpace. We conclude this section with the expressiveness results; we show that classical NWA with forward-walking slave automata and NWA with backward-walking slave automaton have incomparable expressive power. Hence, (finite-width) (LimAvg; Sum$^+$)-automata with bidirectional slave automata are strictly more expressive than NWA with one-direction slave automata.

5.1 The finite-width condition

Finite width. An NWA $\mathbb{A}$ has finite width if and only if in every accepting run of $\mathbb{A}$ at every position at most finitely many slave automata are active. Classical NWA with forward-walking slave automata only have finite width. Indeed, in any run, at any position i at most i slave automata can be active.

▶ **Example 6.** Consider an NWA over $\{a, b\}$ such that the master automaton accepts a single word ab^ω and all slave automata are backward walking and accept words b^*a. All slave automata terminate at the first position of ab^ω and hence this NWA does not have finite width.

The automata expressing properties from Examples 1, 3 and 4 are finite-width (LimAvg; Sum$^+$)-automata with bidirectional slave automata. Observe that an NWA does not have finite width if and only if it has an accepting run, in which at some position i infinitely many backward-walking slave automata terminate.

▶ Remark (Finite width is natural for positive sum). Let $\mathbb{A}$ be a (LimAvg; Sum$^+$)-automaton with bidirectional slave automata. Except for degenerate cases, runs of $\mathbb{A}$, which do not have finite width, have infinite value. Indeed, consider a run π and a position i_0 at which infinitely many automata are active. Since only finitely many forward-walking slave automata are active at i_0, infinitely many of them are backward-walking and for some position $i < i_0$, infinitely many slave automata S terminate at position i. Then, one of the following holds: either that value of this run is infinite or one of the following two degenerate cases happen: (a) The slave automata from S are invoked with zero density (i.e., if consider the longrun-average of the frequency of invoking slave automata, then it is zero). This situation represents that monitoring with slave automata happens with vanishing frequency which is a degenerate case. (b) The values returned by the slave automata from S are bounded. It follows that these automata take transitions of non-zero weight only in some finite subword $w[i, j]$ of the input word w. This situation represents monitoring of an infinite sequence, in which all events past position j are irrelevant. This is a degenerate case in the infinite-word case.

The finite-width property does not depend on weights and hence we can construct an exponential-size Büchi automaton $\mathcal{A}$, which simulates runs of a given NWA $\mathbb{A}$. Having $\mathcal{A}$, we can check whether it has a run corresponding to an accepting run of $\mathbb{A}$, in which infinitely many backward-walking slave automata terminate at the same position. This check can be done in logarithmic space and hence checking the finite-width property is in PSpace.

A simple reduction from the non-emptiness problem for NWA shows PSPACE-hardness of checking the finite-width property.

▶ **Theorem 7.** *The problem whether a given NWA has finite width is* PSPACE*-complete.*

5.2 The absolute sum value function

We present the main result on NWA of finite width.

▶ **Theorem 8.** *The emptiness problem for finite-width* (LIMAVG; SUM$^+$)*-automata with bidirectional slave automata is* PSPACE*-hard and it is decidable in* EXPSPACE*.*

Key ideas. PSPACE-hardness follows from PSPACE-hardness of the emptiness problem for (LIMAVG; SUM$^+$)-automata with forward-walking slave automata only. Containment in EXPSPACE is shown by reduction to the bounded-width case, which is shown decidable in the following section (Theorem 13). We briefly describe this reduction. Consider a finite-width (LIMAVG; SUM$^+$)-automaton $\mathbb{A}$ with bidirectional slave automata. First, we observe that without loss of generality, we can assume that $\mathbb{A}$ is deterministic. Second, we observe that in every word accepted by $\mathbb{A}$, at almost every position i there exists a *barrier*, which is a word u such that (a) the word $w' = w[1,i]uw[i+1,\infty]$, i.e., w with u inserted at position i, is accepted by $\mathbb{A}$, and the runs on w and w' coincide except for positions in w' corresponding to u, (b) in the run on w', backward-walking slave automata active at the end of u terminate within u, (c) in the run on w', forward-walking slave automata active at the beginning of u terminate within u, and (d) u has exponential length. Basically, active slave automata cannot cross u in w' and in the effect insertion of u bounds the number of active slave automata. Existence of barriers follows from the finite-width property of $\mathbb{A}$.

We insert barriers in w to reduce the number of active slave automata. We show that if at position i in w, exponentially many active slave automata accumulates exponential weight past crossing i (some slave automata walk forward while other backwards), all partial averages (of values returned by slave automata) in w' are bounded by the corresponding partial averages in w. We conclude that for every word w, there exists a word w' such that (i) at every position at most exponentially many slave automata accumulate at least exponential values, and (ii) the value of w' does not exceed the value of w. Thus, to compute the infimum over all runs of $\mathbb{A}$, we can focus on runs in which at every position at most exponentially many slave automata accumulate at least exponential values. Runs of slave automata in which they accumulate bounded (exponential) values can be eliminated as in Theorem 5, i.e., we can construct an exponential size NWA $\mathbb{A}'$, which simulates $\mathbb{A}$, and such that its slave automata run as long as they can accumulate value exponential (in $|\mathbb{A}|$) and otherwise they non-deterministically pick the remaining value and the master automaton verifies that the pick is correct. Therefore, the infimum over all runs of $\mathbb{A}$ coincides with the infimum over all runs of $\mathbb{A}'$ of width exponentially bounded.

▶ Remark (Parametrized complexity). If we assume that the size of slave automata in Theorem 8 is bounded by a constant, the complexity of the emptiness problem drops to NLOGSPACE-complete. NLOGSPACE-hardness follows from NLOGSPACE-hardness of the emptiness problem for LIMAVG-automata, which can be viewed as a special case of NWA.

5.3 Expressive power

DCP defined in Example 3 can be expressed by a deterministic finite-width (LIMAVG; SUM$^+$)-automaton with bidirectional slave automata. We show that both forward-walking and

backward-walking slave automata are required to express DCP. That is, we formally show that DCP cannot be expressed by any (non-deterministic) (LimAvg; Sum$^+$)-automaton with slave automata walking in one direction only.

Classes of NWA. We define $\mathcal{FB}$(LimAvg; Sum$^+$) as the class of all finite-width (LimAvg; Sum$^+$)-automata with bidirectional slave automata. We define $\mathcal{F}$(LimAvg; Sum$^+$) (resp., $\mathcal{B}$(LimAvg; Sum$^+$)) as the subclass of $\mathcal{FB}$(LimAvg; Sum$^+$) consisting of NWA with forward-walking (resp., backward-walking) slave automata only.

We establish that classes $\mathcal{F}$(LimAvg; Sum$^+$) and $\mathcal{B}$(LimAvg; Sum$^+$) have incomparable expressive power and hence they are strictly less expressive than class $\mathcal{FB}$(LimAvg; Sum$^+$).

Key ideas. Consider a word $w = (c\#^N r^{2K} c\#^{2N} r^K)^\omega$ for some big K and much bigger N. An NWA from $\mathcal{B}$(LimAvg; Sum$^+$) computes DCP of w by invoking (non-dummy) slave automata at every r letter and taking silent transitions on letters $\#, c$. We show that an NWA $\mathbb{A}$ from $\mathcal{F}$(LimAvg; Sum$^+$) cannot invoke the right number of slave automata, even if it uses non-determinism. More precisely, we show that $\mathbb{A}$ computing DCP has to invoke at most $O(K)$ non-dummy slave automata on average on subwords $c\#^N r^{2K} c\#^{2N} r^K$. Since N is much bigger than K, we conclude that $\mathbb{A}$ has a cycle over $\#$ letters at which it takes only silent transitions and a cycle over r letters on which it increases the multiplicity of active slave automata. Using these two cycles, we construct a run of value smaller than DCP, which contradicts the assumption that $\mathbb{A}$ computes DCP. Similarly, we can show that an NWA from $\mathcal{B}$(LimAvg; Sum$^+$) cannot compute correctly DCP of words of the form $w = (cw^{2K}\#^N cw^K\#^{2N})^\omega$, while on these words DCP is expressible by an NWA from $\mathcal{F}$(LimAvg; Sum$^+$).

▶ **Lemma 9.** *(1) DCP restricted to alphabet $\{r, \#, c\}$ is expressed by an NWA from $\mathcal{B}$(LimAvg; Sum$^+$), but it is not expressible by NWA from $\mathcal{F}$(LimAvg; Sum$^+$). (2) DCP restricted to alphabet $\{w, \#, c\}$ is expressed by an NWA from $\mathcal{F}$(LimAvg; Sum$^+$), but it is not expressible by NWA from $\mathcal{B}$(LimAvg; Sum$^+$).*

The above lemma implies that DCP over alphabet $\{r, w, \#, c\}$ is not expressible by any NWA from $\mathcal{F}$(LimAvg; Sum$^+$) nor from $\mathcal{B}$(LimAvg; Sum$^+$). In conclusion, we have:

▶ **Theorem 10.** *(1) $\mathcal{F}$(LimAvg; Sum$^+$) and $\mathcal{B}$(LimAvg; Sum$^+$) have incomparable expressive power. (2) $\mathcal{FB}$(LimAvg; Sum$^+$) are strictly more expressive than $\mathcal{F}$(LimAvg; Sum$^+$) and $\mathcal{B}$(LimAvg; Sum$^+$).*

6 Bounded-width case

In this section, we study (LimAvg; Sum)-automata with bidirectional slave automata, which have bounded width. The bounded width restriction has been introduced in [14] to improve the complexity of the emptiness problem and to establish decidability of the emptiness problem for (LimAvg; Sum)-automata. NWA considered in [14] have only forward-walking slave automata, while we extend these results to NWA with bidirectional slave automata. This extension preserves the complexity bounds from [14], i.e., the emptiness problem is in PTime for constant width and PSpace-complete for width given in unary.

The bounded width restriction emerges naturally in examples presented so far. If we bound the number of pending requests, we can express ART and AW (Example 1) by automata of bounded width. If we bound the number of writes and reads between any two

commits, then DCP and MR (Examples 3 and 4) can be expressed by NWA of bounded width. These natural restrictions lead to more efficient decision procedures.

The decision procedure in this section differs from the one from [14]. The key step in the decidability proof from [14] is establishing the following dichotomy: either the infimum over values of all words is $-\infty$ or the infimum is realized by *dense* runs. A run is dense if for the values $v_1, v_2, \ldots$ returned by slave automata invoked at positions $1, 2, \ldots$ we have $\frac{v_i}{i}$ converges to 0, i.e., values returned by slave automata are sublinear in the positions of their invocation. Properties of dense runs allow for further reductions, which lead to a decision procedure. However, we show in the following Example 11 that for NWA with bidirectional slave automata, dense runs may not attain the infimum of all runs.

▶ **Example 11.** Consider a (LimAvg; Sum)-automaton $\mathbb{A}$ with bidirectional slave automata over $\Sigma = \{a, b, c\}$. The NWA $\mathbb{A}$ accepts words $(ab^*c)^\omega$ and it works as follows. On letters a, $\mathbb{A}$ invokes a forward-walking slave automaton $\mathfrak{B}_a$, which returns the number of the following b letters up to c. On letters c, $\mathbb{A}$ invokes a backward-walking slave automaton $\mathfrak{B}_c$, which returns the number of the preceding b letters since a. Finally, on b letters, $\mathbb{A}$ invokes a slave automaton $\mathfrak{B}_b$, which takes a single transition and returns value 0. The NWA $\mathbb{A}$ has width 3. We can show that the value of any dense run, is 2. However, the infimum over values of all words is 1. The partial average of the values returned by slave automata on finite word $u = (ab^*c)^*$ is 2, while the partial average over uab^N is $\frac{2|u|+N}{|u|+N}$. Therefore, the value, which is limit infimum over partial averages, of word $ab^{n_1}c \ldots ab^{n_i}c \ldots$ is 1 if sequence $n_1, n_2, \ldots$ grows rapidly (e.g. doubly-exponentially).

Main ideas. In Example 11, the words attaining the infimum contain long blocks of letter b, at which the NWA $\mathbb{A}$ is (virtually) in the same state, i.e., it loops in this state. On letters b, the sum of all weights collected by all active slave automata is 2, i.e., automata $\mathfrak{B}_a, \mathfrak{B}_c$ collect weight 1 and $\mathfrak{B}_b$ collect 0. However, in computing limit infimum over partial averages, we pick positions just before letter c as they correspond to the local minima, i.e., we compute the partial average over prefixes uab^N, and hence the weights collected by $\mathfrak{B}_c$ do not contribute to this partial average. Then, the sum of all weights collected by slave automata $\mathfrak{B}_a, \mathfrak{B}_b$ over a letter b is 1, which is equal to the least value of the limit infimum of the partial averages. In the following, we extend this idea and present the solution for all bounded-width (LimAvg; Sum)-automata with bidirectional slave automata. We show that the infimum over all words of a given NWA is the least average value over all cycles. In the following, we define appropriate notions of cycles of NWA and their average with exclusion of some slave automata.

The graph of k-configurations. Let $\mathbb{A}$ be a non-deterministic (LimAvg; Sum)-automaton of width k. We define a *k-configuration* of $\mathbb{A}$ as a tuple $(q; q_1, \ldots, q_k)$ where q is a state of the master automaton, and each $q_1, \ldots, q_k$ is either a state of a slave automaton of $\mathbb{A}$ or $\perp$. Given a run of $\mathbb{A}$, we say that $(q; q_1, \ldots, q_k)$ is the k-configuration at position i in the run if q is the state of the master automaton at position i and there are $l \leq k$ active slave automata at position i, whose states are $q_1, \ldots, q_l$ ordered by position of invocation (backward-walking slave automata are invoked past position i). If $l < k$, then $q_{l+1}, \ldots, q_k = \perp$. We say that a k-configuration C_2 is a successor of a k-configuration C_1 if there exists an accepting run of $\mathbb{A}$ and $i > 0$ such that C_1 is the k-configuration at i and C_2 is the k-configuration at $i + 1$. The *graph of k-configurations* of $\mathbb{A}$ is the set of k-configurations of $\mathbb{A}$, which occur infinitely often in some accepting run, with the successor relation.

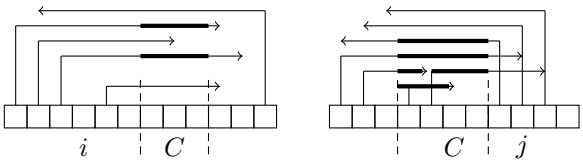

Figure 2 Pictorial explanation of $\textsc{Gain}(\mathcal{C}, FC)$ (on the left) and $\textsc{AvgE}(\mathcal{C}, R)$ (on the right). The gain $\textsc{Gain}(\mathcal{C}, FC)$ on the left is the sum of weights corresponding to thick parts of runs of slave automata invoked before i. The average $\textsc{AvgE}(\mathcal{C}, R)$ corresponds to the average of the thick parts of runs divided by the number of slave automata invoked within C. Slave automata invoked past j are excluded from the average.

Characteristics of cycles. Let $\mathcal{C}$ be a cycle in a graph of k-configurations of $\mathbb{A}$. Let F (resp., B) be the set of forward-walking (resp., backward-walking) slave automata, which are active throughout $\mathcal{C}$, i.e., which are not invoked nor terminated within $\mathcal{C}$. A *focus* Fc (for $\mathcal{C}$) is a downward closed subset of F, i.e., it contains all automata from F invoked before some position. We define a focused gain $\textsc{Gain}(\mathcal{C}, Fc)$ as the sum of weights which automata from Fc accumulate over $\mathcal{C}$. A *restriction* R (for $\mathcal{C}$) is an upward closed subset of B, i.e., it contains all automata from B invoked past certain position. We define an average weight of $\mathcal{C}$ excluding R, denoted by $\textsc{AvgE}(\mathcal{C}, R)$, as the sum of weights of all transitions of slave automata within $\mathcal{C}$, except of transitions of slave automata from R, divided by the number of slave automata invoked within $\mathcal{C}$.

Intuitively, a focused gain refers to the value, which forward-walking slave automata invoked before some position i, accumulate over the part of run corresponding to $\mathcal{C}$ (see Figure 2). If the focused gain $\textsc{Gain}(\mathcal{C}, Fc)$ is negative, then by pumping $\mathcal{C}$ we can arbitrarily decrease the partial average of the values of slave automata invoked before i. In consequence, we can construct a run of the value $-\infty$. Formally, we define condition (*), which implies that there exists a run of value $-\infty$, as follows: (*) there exists a cycle $\mathcal{C}$ in the graph of k-configurations of $\mathbb{A}$ and a focus Fc such that $\textsc{Gain}(\mathcal{C}, Fc) < 0$.

If the focused gain of every cycle is non-negative, we need to examine averages of cycles, while excluding some backward-walking slave automata. The average weight with restriction corresponds to the partial average of values aggregated over $\mathcal{C}$ by all slave automata invoked before position j (which can be past $\mathcal{C}$). Backward-walking slave automata in the restriction correspond to automata invoked past j, and hence their values do not contribute to the partial average (up to i) (see Figure 2). In Example 11, we compute the average of slave automata over letters b, but we exclude the backward-walking slave automaton invoked at the following letter c. Observe that for any cycle $\mathcal{C}$ and any restriction R, having a run containing $\mathcal{C}$ occurring infinitely often, we can repeat each occurrence of cycle $\mathcal{C}$ sufficiently many times so that the partial average of values of slave automata up to position corresponding to j becomes arbitrarily close to the average $\textsc{AvgE}(\mathcal{C}, R)$. The resulting run contains a subsequence of partial averages convergent to $\textsc{AvgE}(\mathcal{C}, R)$ and hence its value does not exceed $\textsc{AvgE}(\mathcal{C}, R)$. We can now state our key technical lemma. This lemma is a direct extension of an intuition behind computing the infimum over values of all words of the NWA $\mathbb{A}$ from Example 11.

▶ **Lemma 12.** *Let $\mathbb{A}$ be a $(\textsc{LimAvg}; \textsc{Sum})$-automaton of bounded width with bidirectional slave automata. (1) If condition (*) holds, then $\mathbb{A}$ has a run of value $-\infty$. (2) If (*) does not hold, then the infimum $\inf_w \mathbb{A}(w)$ equals the infimum $\inf_{\mathcal{C}\in\Lambda,R} \textsc{AvgE}(\mathcal{C}, R)$, where Λ is the set of all cycles $\mathcal{C}$ in the graph of k-configurations of $\mathbb{A}$.*

If the width of $\mathbb{A}$ is constant, then the graph of k-configurations has polynomial size in $|\mathbb{A}|$ and it can be constructed in polynomial time by employing reachability checks on the set

Table 1 The complexity of the emptiness problem for (LimAvg; g)-automata. The columns describe respectively: the value function g, restrictions imposed on the problem, the complexity in the case with bidirectional slave automata, and the complexity in the previously studied [13, 14] case with only forward-walking slave automata. Results presented in this paper are boldfaced.

Value func. g	Restrictions	Complexity Bidirectional	Complexity Forward case
Min,Max, $\text{Sum}^{L,U}$	None	**PSpace-complete** (Thm 5)	PSpace-complete [13]
Sum^{+}	finite width	**PSpace-hard ExpSpace** (Thm 8)	PSpace-hard ExpSpace [13]
Sum^{+}, Sum	constant width unary weights	**NLogSpace-complete** (Thm 13)	NLogSpace-complete [14]
Sum^{+}, Sum	constant width binary weights	**PTime** (Thm 13)	PTime [14]
Sum^{+}, Sum	width given in unary	**PSpace-complete** (Thm 13)	PSpace-complete [14]

of all k-configurations w.r.t. to relaxation of the successor relation. Therefore, for every focus Fc and every k-configuration c we can check in polynomial time whether there exists a cycle $\mathcal{C}$ such that $\mathcal{C}[1] = c$ and $\text{Gain}(\mathcal{C}, Fc) < 0$. Thus, condition (1) can be check in logarithmic space assuming that weights are given in unary. If weights are given in binary, condition (1) can be checked in polynomial time. Checking condition (2) has the same complexity as condition (1). If the width k is given in unary in input, the graph of k-configurations is exponential in $|\mathbb{A}|$ and conditions (1) and (2) can be checked in polynomial space. Weights in this case are logarithmic in the size of the graph and hence changing representation from binary to unary does not affect the (asymptotic) size of the graph.

▶ **Theorem 13.** *The emptiness problem for* (LimAvg; Sum)*-automaton of width k with bidirectional slave automata is (a)* NLogSpace*-complete for constant k and weights given in unary, (b) in* PTime *for constant k and weights given in binary, and (c)* PSpace*-complete for k given in unary.*

7 Discussion and Conclusion

Discussion. We established decidability of the emptiness problem for classes of bidirectional NWA, which include all NWA presented in the examples. An NWA from Example 2 is covered by Theorem 5, while NWA from Examples 1, 3 and 4 are covered by Theorem 8. The lower bounds in our results follow from the lower bounds of the special case of forward-only NWA. The established complexity (Table 1) coincide with the forward-only case.

Concluding remarks. In this work we present bidirectional NWA as a specification formalism for quantitative properties. There are several interesting directions for future work. The study of bidirectional NWA with other value functions is an interesting direction. The second direction of future work is to consider other formalism (such as a logical framework) which has the same expressive power as bidirectional NWA.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In *TACAS, 2014*, pages 424–439, 2014.

2 Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *J. ACM*, 63(3):24:1–24:56, 2016. doi:10.1145/2875421.

3 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS 2013*, pages 13–22, 2013.

4 Mikołaj Bojańczyk and Thomas Colcombet. Bounds in w-regularity. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 285–296, 2006. doi:10.1109/LICS.2006.17.

5 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM TOCL*, 15(4):27:1–27:25, 2014.

6 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. In *ICALP 2010, Part II*, pages 587–598. Springer, 2010.

7 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR 2014*, pages 266–280, 2014.

8 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim Guldstrand Larsen, and Simon Laursen. Average-energy games. In *GandALF 2015.*, pages 1–15, 2015. doi:10.4204/EPTCS.193.1.

9 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In *CONCUR*, pages 269–283, 2010. doi:10.1007/978-3-642-15375-4_19.

10 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *FCT'09*, pages 3–13. Springer, 2009.

11 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *LMCS*, 6(3), 2010.

12 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.

13 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *LICS 2015*, pages 725–737, 2015.

14 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted limit-average automata of bounded width. In *MFCS 2016*, pages 24:1–24:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.24.

15 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative automata under probabilistic semantics. In *LICS 2016*, pages 76–85, 2016. doi:10.1145/2933575.2933588.

16 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative monitor automata. In *SAS 2016*, pages 23–38, 2016. doi:10.1007/978-3-662-53413-7_2.

17 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Bidirectional nested weighted automata. *CoRR*, abs/1706.08316, 2017. URL: http://arxiv.org/abs/1706.08316.

18 Krishnendu Chatterjee and Vinayak S. Prabhu. Quantitative temporal simulation and refinement distances for timed systems. *IEEE Trans. Automat. Contr.*, 60(9):2291–2306, 2015. doi:10.1109/TAC.2015.2404612.

19 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

20 Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In *DLT 2006*, pages 49–58, 2006.

21 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. In *STACS, 2016*, pages 53:1–53:13, 2016. doi:10.4230/LIPIcs.STACS.2016.53.

22 Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Aut. Lang. & Comb.*, 7(3):321–350, 2002.